Zum Lernverhalten von Backpropagation–Netzen auf der Basis stochastischer Rechentechnik

Vom Fachbereich Elektrotechnik
der Universität der Bundeswehr Hamburg
zur Erlangung des akademischen Grades eines
Doktor-Ingenieurs genehmigte

DISSERTATION

vorgelegt von
M. Sc. Liyun Zhu
aus Sichuan, V.R. China

Hamburg 18. Juni 2000

Gutachter:	Prof. Dr.-Ing. H.Ch. Zeidler Prof. Dr. rer. nat. H. Homuth
Vorsitzender der Prüfungskommission:	Prof. Dr.-Ing. H. Göbel
Mündliche Prüfung:	03. August 2000

Gedruckt mit Unterstützung der Universität der Bundeswehr Hamburg

REPORT DOCUMENTATION PAGE

Form Approved OMB No. 0704-0188

1. AGENCY USE ONLY (Leave blank)	2. REPORT DATE	3. REPORT TYPE AND DATES COVERED
	2000	Dissertation

4. TITLE AND SUBTITLE

Zum Lernverhalten von Backpropagation- Netzen auf der Basis strochastischer Rechentechnik

On the Learning Process of Back-Propagation Nets on the Basis of Stochastic Computation

5. FUNDING NUMBERS

6. AUTHOR(S)

Liyun Yu

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)

Fakultaet fuer Elektrotechnik, Universitaet der Bundeswehr Hamburg

8. PERFORMING ORGANIZATION Report Number REPORT NUMBER

9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES

10. SPONSORING/MONITORING AGENCY REPORT NUMBER

11. SUPPLEMENTARY NOTES

Text in German. Title and abstract in German and English, 140 pages.

12a. DISTRIBUTION/AVAILABILITY STATEMENT

Distribution A: Public Release.

12b. DISTRIBUTION CODE

ABSTRACT (Maximum 200 words)

The dissertation entails a detailed examination of the adaptive or learning process of a back-propagation net, used for the building of massive parallel neural networks, and based on the level of mathematical induction, stochastic computation, and software simulation. Firstly are the introductory bases of the back-propagation new discussed. Parameters and an adaptive process are then selected, followed by limitations and countermeasures. The results of the examination and software simulation demonstrate the simultaneously occurring work and learning processes do not seem to exert any negative influence on the convergence progress. The more precise and quantitative scalability of the entire process is evaluated in the study's final chapter.

14. SUBJECT TERMS

German, UNIBW, Neural networks, Back-propagation net, Software simulation, Learning/adaptive processes, Stochastic computation

15. NUMBER OF PAGES

16. PRICE CODE

17. SECURITY CLASSIFICATION OF REPORT	18. SECURITY CLASSIFICATION OF THIS PAGE	19. SECURITY CLASSIFICATION OF ABSTRACT	20. LIMITATION OF ABSTRACT
UNCLASSIFIED	UNCLASSIFIED	UNCLASSIFIED	UNLIMITED

Zum Lernverhalten von Backpropagation–Netzen auf der Basis stochastischer Rechentechnik

Vom Fachbereich Elektrotechnik
der Universität der Bundeswehr Hamburg
zur Erlangung des akademischen Grades eines
Doktor-Ingenieurs genehmigte

DISSERTATION

vorgelegt von
M. Sc. Liyun Zhu
aus Sichuan, V.R. China

Hamburg 18. Juni 2000

Gutachter:	Prof. Dr.-Ing. H.Ch. Zeidler
	Prof. Dr. rer. nat. H. Homuth
Vorsitzender der Prüfungskommission:	Prof. Dr.-Ing. H. Göbel
Mündliche Prüfung:	03. August 2000

Gedruckt mit Unterstützung der Universität der Bundeswehr Hamburg

Danksagung

Ein einzelnes Neuron hat kaum Bedeutung und leistet fast nichts. Aber ein neuronales Netz, welches aus mehr als tausend oder hunderttausend Neuronen besteht, kann uns Menschen dadurch überraschen, was es schaffen kann. Dies ist ein Naturgesetz und trifft ebenfalls auf Menschen zu. In dem Sinne möchte ich mich an dieser Stelle ganz herzlich bei denen bedanken, die mir den Weg zum Erscheinen dieses Werks geebnet haben. Ohne deren Unterstützung wäre es kaum vorstellbar, daß die vorliegende Arbeit zustande gekommen und veröffentlicht worden wäre.

An aller erster Stelle möchte ich mich bei Prof. Dr.-Ing. H. Ch. Zeidler für die wertvollen Anregungen und die Betreuung dieser Arbeit sowie die vertrauensvolle Unterstützung bedanken. Seine Offenheit, sein Interesse und Engagement hat die Umsetzung meiner Ideen ermöglicht. Herrn Prof. Dr. rer. nat. H. Homuth danke ich für die Übernahme des Korreferats und für die gelegentlichen interessanten wissenschaftlichen Erörterungen. Herrn Prof. Dr.-Ing. H. Göbel danke ich für die Übernahme des Vorsitzes der Prüfungskommission.

Weiterer Dank geht an Herrn Dr. rer. nat. K. Köllmann, der mich durch viele anregende Diskussionen zu neuen Ideen inspiriert und beim Korrekturlesen dieser Arbeit wertvolle Hinweise gegeben hat. Gedankt sei auch Herrn Dr. Rübel für die wertvolle Diskussion, welche die damals steckengebliebene Arbeit wieder ermunterte. Mein besonderes Dankeschön geht an Herrn Dr.-Ing. K.-R. Riemschneider, dessen Vorarbeit und Engagement den Ausgangspunkt der vorliegenden Arbeit gebildet und in der Anfangsphase dieser Arbeit die Richtung der Weiterentwicklung mitbestimmt hat. Zum Schluß geht mein Dank natürlich an alle meine Kollegen für die äußerst angenehme und hilfsbereite Arbeitsatmosphäre und wertvolle Unterstützung der Arbeit.

Hamburg, im Dezember 2000 Liyun Zhu

Kurzfassung

Die Verwendung von Neurochips mit der Fähigkeit des *On–chip–learning* stellt eine Lösung dar für den Aufbau massiv paralleler Neurosysteme. Bei der Implementierung eines solchen Chips kann analoge oder digitale Technik verwendet werden, die aber unterschiedliche Vor– und Nachteile aufweist. Ein auf der stochastischen Rechentechnik basiertes Verfahren wurde von Riemschneider u.a. vorgeschlagen [51], welches die Vorteile der beiden Techniken ausnutzt und den bekannten Backpropagation–Algorithmus als Lernregel verwendet. Über die Auswahl der Netzparameter und die Konvergenzeigenschaften bei großen Netzen wurden bisher jedoch sehr wenig Aussagen getroffen. Deshalb wird in der vorliegenden Arbeit eine ausführliche Untersuchung zum Lernverhalten der durch das vorgeschlagene Verfahren implementierten Netze durchgeführt. Die Untersuchung basiert auf der Ebene der mathematischen Herleitung und Software–Simulation. Eine 1:1–Abbildung der Netzparameter zu denen der konventionellen BP–Verfahren wird mathematisch hergeleitet. Dadurch werden Unter– und Obergrenzen der Netzparameter abgeschätzt und mit Beispielen per Software–Simulation bestätigt. Die potentiell einschränkenden Einflußgrößen des vorgestellten Verfahrens werden dann gründlich studiert und anschließend werden entsprechende Gegenmaßnahmen vorgeschlagen. Den Schwerpunkt bildet die Einführung eines modifizierten Neurons, welches sich leicht in das vorhandene Verfahren integrieren läßt und dessen Ausgangssignal ebenfalls einer S–förmigen Funktion entspricht. Das modifizierte Neuron ist in vieler Hinsicht, z.B. bessere Konvergenzeigenschaften beim Training, dem Vorgänger überlegen. Dieser Ansatz wird dann durch eine Software–Simulation mit unterschiedlichen Beispielen bestätigt.

Hinsichtlich der Ergebnisse der Untersuchung und der Software–Simulation wird festgestellt, daß der gleichzeitig auftretende Arbeits– und Lernvorgang (das sogenannte Gegenstrom–Verfahren) keine negativen Einflüsse auf die Konvergenzeigenschaften des vorgestellten Verfahrens erkennen läßt. Die Verwendung der mittelnden Addition statt einer arithmetischen Addition für die Summation der gewichteten Neuroneneingänge (das sogenannte $\frac{1}{N}$–Mittelungsverfahren) bringt ebenfalls keine Schwierigkeit für die Konvergenz des Verfahrens beim Training, sofern eine steilere S–Funktion eingesetzt wird.

Zum Schluß wird lediglich eine sehr grobe Abschätzung zur Skalierbarkeit des vorgestellten Verfahrens getroffen. Jedoch wird ein möglicher Weg zu einer genaueren und quantitativen Aussage der Skalierbarkeit aufgezeigt.

Inhaltsverzeichnis

1 Einleitung 1

1.1 Motivation 1

1.2 Zielsetzung 3

1.3 Aufbau der Arbeit 5

2 Einführende Grundlagen zu Backpropagation–Netzen 7

2.1 Netzaufbau 7

2.1.1 Multilayer Netz (MLN) 8

2.1.2 Anzahl der Neuronen in einem MLN 8

2.1.2.1 Ein– und Ausgangsschichten 8

2.1.2.2 Verborgene Schichten 9

2.1.3 Verbindung zwischen Schichten 9

2.1.3.1 Vollverbindung 10

2.1.3.2 Teilverbindung 11

2.2 Lernalgorithmus 12

2.2.1 Warum Backpropagation? 12

2.2.2 Lernen mit Backpropagation (BP) 13

2.2.2.1 Problemstellung 13

2.2.2.2 Arithmetische Operationen 14

2.2.2.3 Ablauf des Trainings 15

2.2.3 Varianten des BP 16

2.3 Umsetzung in stochastische Rechentechnik 17

2.3.1 Codierung und Decodierung 18

2.3.1.1 Implementierung 18

2.3.1.2 Stochastische Streuung 19

2.3.2 Arithmetische Operationen 21

2.3.2.1 Multiplikation 21

2.3.2.2 Addition 21

2.3.2.3 Subtraktion 22

2.3.3 Die nichtlineare Überführungsfunktion 23

3 Parameterwahl und Lernverhalten 29
3.1 Gewichte 29
3.1.1 INDIE–Glied 30
3.1.2 ADDIE–Glied 31
3.1.3 Synapsenelement 32
3.2 Das Verhältnis zum konventionellen BP–Algorithmus 32
3.2.1 Eine 1:1–Abbildung 33
3.2.2 Vorschläge zur Parameterwahl 35
3.3 Parameterwahl bei taktgenauer Bitstrom–Simulation 36
3.3.1 ADDIE–Länge 37
3.3.2 INDIE–Länge 40
3.3.3 Taktanzahl je Präsentation 44
3.3.4 Runlänge der stochastischen Automaten 46

4 Einschränkungen und Gegenmaßnahmen 49
4.1 Unterschiede zum konventionellen BP–Algorithmus 49
4.2 Auswirkungen auf die Konvergenz 50
4.2.1 BP–Netz in konventioneller Rechentechnik 51
4.2.1.1 [0,1]–Einschränkung 51
4.2.1.2 $[\frac{1}{N}]$–Problem 53
4.2.1.3 Squash– und Bogenfunktion 55
4.2.1.4 Zusammenwirken der Einschränkungen . . . 59
4.2.2 BP–Netz in stochastischer Rechentechnik 60
4.2.2.1 Einsatz von S– und B–Funktion 62
4.2.2.2 Ausbreitung der stochastischen Streuung . . 67
4.2.2.3 Gegenstrom–Verfahren 73
4.3 Gegenmaßnahmen 75
4.3.1 *Online–Training* statt Gegenstrom–Verfahren 75
4.3.2 ADDIE als Tiefpaßfilter zwischen Neuronenschichten . 77
4.3.3 Einsatz eines modifizierten Neurons 84
4.3.3.1 Theoretisches Modell und Kennlinie 84
4.3.3.2 Implementierung des modifizierten Neurons . 85
4.3.3.3 Ableitung der neuen S–Funktion 88
4.3.3.4 Analyse und Experiment 91
4.3.4 Kurze Diskussion 94
4.4 Beurteilung und Testbeispiele 96

4.4.1 Behandelte Problemklassen und Beispiele 98
4.4.1.1 Das Parity–Problem 100
4.4.1.2 BEK über Bitmap–Repräsentation 100
4.4.1.3 BEK via numerische Merkmale 101
4.4.2 Ergebnisse und Diskussion 102
4.4.2.1 Gegenstrom–Verfahren 102
4.4.2.2 Verbesserte Konvergenzeigenschaften durch modifizierte Neuronen 103
4.4.2.3 Skalierbarkeit 106
4.4.2.4 Nichtbinäre Aufgabenstellungen 107
4.5 Schlußbemerkung 111

5 Zusammenfassung und Perspektiven 115
5.1 Zusammenfassung 115
5.2 Perspektiven 116

Literaturverzeichnis 119

A Simulationssoftware 123
A.1 Aufbau des Programms 123
A.1.1 Zufallsquelle 123
A.1.2 Neuronen und Synapsen 125
A.2 Schnittstelle zum Benutzer und Ablauf des Programms 126
A.2.1 Vorbereitung der Trainingsmuster–Datei 127
A.2.2 Programmablauf 127

B Gesetze der großen Zahl und Skalierbarkeit 131
B.1 Gesetze der großen Zahl 131
B.2 Diskussion 132

C Mehrfach verwendete Formelzeichen und Abkürzungen 135

1 Einleitung

1.1 Motivation

Das menschliche Gehirn ist seit langer Zeit ein faszinierendes Forschungsgebiet. Viele Wissenschaftler aus fast allen Disziplinen machen unermüdlich große Anstrengungen, um seine Arbeitsweise zu verstehen mit dem Ziel, daß man es teilweise nachbilden kann. Neuronale Biologen haben angefangen, die Reizreaktion einzelner Neuronen auf der physiologischen Ebene zu erforschen, während sich Psychologen die Gehirnfunktionen auf der kognitiven und verhaltensorientierten Ebene erarbeiten. Man hofft, daß sich die beiden Gruppen eines Tages irgendwo auf dem Forschungsweg treffen, so daß die Rätsel des menschlichen Gehirns gelöst werden können. Aber bis dahin liegt noch ein langer Weg vor uns. Elektrotechnische Ingenieure und Informatiker wollen und können nicht so lange warten. Sie versuchen ständig mit Hilfe der bisherigen Ergebnisse und ebenfalls mit Unterstützung der Mathematiker, die wesentliche Funktion des menschlichen Gehirns, nämlich die Lernfähigkeit des Menschen, nachzubilden oder zu simulieren. Dafür wurden Software– und Hardwaresysteme entwickelt, die künstliche Neuronale Netze (ANN oder NN als Kurzform) genannt werden.

Verglichen mit konventioneller Datenverarbeitung ist die Lernfähigkeit die entscheidende Eigenschaft eines NN. Der Prozessor eines Computers kann als ein einzelnes Neuron betrachtet werden, der Informationen bearbeiten kann. Aus diesem Blickfeld ist die Softwarerealisierung eines NN auf einem konventionellen Computer keine gute Lösung, weil Informationen, im Gegensatz zur massiv parallelen Informationsverarbeitung im menschlichen Gehirn, nur seriell bearbeitet werden können. Die Softwarerealisierung auf Rechnern mit Multiprozessoren hat einerseits die Schwierigkeit der Kommunikation zwischen Neuronen, wenn die Anzahl der Prozessoren relativ groß ist, anderseits ist die Komplexität der heutigen parallelen Multiprozessorsysteme nicht vergleichbar mit der Neuronenanzahl des menschlichen Gehirns (ca. 10^{10}). So versuchen Naturwissenschaftler und Ingenieure, spezielle Hardware für NNs aufzubauen, welche die wesenseigene Parallelität neuronaler Methoden nutzen kann.

Der Aufbau von Neurochips mit der Fähigkeit des *On–chip–learning* könnte eine Lösung sein, um obigen Erwartungen in Richtung massiv paralleler Systeme näher zu kommen. Man hofft, daß man durch solche Neurochips große neuronale Netze leicht aufbauen kann, um den Einsatz von

NNs weiter voranzutreiben. Bei der Implementierung eines solchen Chips kann analoge oder digitale Technik verwendet werden, die aber unterschiedliche Vor- und Nachteile aufweist. Eine analoge Implementierung benötigt weniger Chipfläche und besitzt mehr Ähnlichkeiten mit biologischen neuronalen Netzen. Ein solches Chip könnte dann mit höherer Geschwindigkeit betrieben werden. Aber die begrenzte Genauigkeit ist ein großes Hindernis für den Einsatz mancher Trainingsalgorithmen (z.B. dem Backpropagation-Algorithmus, der eine bestimmte Mindestgenauigkeit zum Rechnen erfordert, siehe [44]). Schwierigkeiten beim Speichern der Gewichte, Empfindlichkeit auf Umgebungseinflüsse (z.B. Rauschen, Temperatur, usw.) und ein relativ komplizierter Entwurfsvorgang müssen in Kauf genommen werden. Maßnahmen gegen solche Nachteile bedeuten oft mehr Platzbedarf auf dem Neurochip, so daß die Vorteile sich praktisch nicht nutzen lassen. Bei der digitalen Implementierung dagegen kann höhere Genauigkeit erreicht und beim Entwurf können vor allem *state-of-the-art* Techniken mit VLSI und ULSI verwendet werden. Dadurch wird die Chiprealisierung erheblich vereinfacht. Ihr wesentlicher Nachteil ist der hohe Platzbedarf auf dem Chip. Eine digitale Multiplizierschaltung z.B. nimmt mehr Chipfläche in Anspruch als ihr analoges Pendant [30]. In Anbetracht der obengenannten Umstände sind viele Wissenschaftler mit vollem Einsatz auf der Suche nach einer Kompromißlösung, die die Vorteile von beiden Techniken vereinen könnte. So tauchten sogenannte *hybride* Chips auf, die mit beiden Techniken implementiert wurden [11] [12]. Gegenüber der analogen Technik bevorzugen viele Informatiker und Elektrotechniker den digitalen Konkurrenten wegen seiner Flexibilität und Leichtigkeit beim Entwurf des Chips und seiner höheren Genauigkeit. Ein anderer Ansatz zur Einsparung von Chipfläche versucht die neuronalen Algorithmen so abzuwandeln, daß die Multiplikationsoperation vermieden wird [17] [41].

Der Einsatz von stochastischen Rechenwerken könnte eine potentielle Lösung sein und ist ein interessantes Forschungsgebiet für die digitale Implementierung lernfähiger Neurochips. In der stochastischen Rechentechnik, die in den sechziger Jahren entwickelt und danach vernachlässigt wurde [36], wird jeder Datenwert durch eine binäre Bitfolge repräsentiert, deren Wahrscheinlichkeit für das Auftreten einer Eins dem Datenwert entspricht. Dadurch kann eine Multiplikation zwischen Datenwerten durch eine UND- oder Äquivalenz-Verknüpfung (je nachdem, ob unipolare oder bipolare Codierung verwendet wird) der entsprechenden Bitfolgen realisiert werden [63]. So ist der Aufbau eines Neurochips mit Lernfähigkeit und auf der Basis stochastischer Rechenwerke ein interessantes Thema geworden. Allerdings liegen dafür bisher nur wenige Realisierungsansätze vor. Der erste derartige Chip gründet auf Arbeiten von Tomlinson [65]. In dieser Richtung

wurden später noch einige Vorschläge gemacht [7] [53] [54] (Verwendung einer unipolaren Codierung) [67] [69] und [68] (Verwendung einer bipolaren Codierung), welche die Idee von Tomlinson aufgriffen und verbesserten. Dennoch sind weiterführende Details und Anwendungsbeispiele bisher nicht bekannt. In dieser Situation wurde von Riemschneider ein Verfahren vorgeschlagen, welches eine im Aufwand günstige Hardwareschaltung befähigt, den Trainingsvorgang in neuronalen Netzen der Backpropagation–Klasse *on–chip* auszuführen [51].
Im Unterschied zu vergleichbaren Ansätzen zeichnet sich sein Verfahren durch folgende Eigenschaften aus:

- Eine bipolare Codierung, welche jede Maschinengröße in einen Bitstrom umwandelt und verarbeitet. Dadurch werden die Darstellung und Verarbeitung vorzeichenbehafteter Größen erlaubt. So können nun mit Hilfe von einfachen Schaltungen Multiplikationen, die im Backpropagation–Algorithmus von zentraler Bedeutung sind, direkt und vorzeichenrichtig ausgeführt werden.

- Die Implementierung der notwendigen Nichtlinearitäten und deren Ableitungen durch stochastische Automaten. Die Form der Nichtlinearitäten (Steilheit der sigmoidähnlichen bzw. Höhe der sigmoid–ableitungsähnlichen Funktionen) ist einstellbar und von der Netzstruktur unabhängig, im Gegensatz zu den bisher bekannten einfachen Ansätzen (vorzeichenbehaftete OR–Verknüpfung).

- Im Gegensatz zu den anderen Methoden, die einen vereinfachten und geänderten BP–Algorithmus verwenden, ermöglichen die obigen Punkte, ein Netz von der Backpropagation–Klasse durch einen typischen BP–Algorithmus mit Momentum–Term, der in der Literatur zu NNs als sehr erfolgreich beschrieben wurde, zu implementieren.

- Der BP–Lernmechanismus wurde intern auf dem Neurochip integriert. Dadurch können die Arbeits– und die Lernphase, vergleichbar einem Gegenstromverfahren, gleichzeitig wirksam sein. Auch konnte Riemschneider nachweisen, daß sich die Verknüpfungen zwischen den Neuronen und ihren Bestandteilen so anordnen lassen, daß sich Neuronale Netze variabler Struktur und Größe durch geeignete Kombination und Verschaltung solcher Chips aufbauen lassen.

1.2 Zielsetzung

Ziel der folgenden Arbeit ist es, das in [51] vorgeschlagene Verfahren einer erweiterten wissenschaftlichen Analyse, Bewertung und einer möglichen

Verbesserung zu unterziehen, dies insbesondere im Hinblick darauf, ob und auf welche Weise das Verfahren skalierbar ist und sich dann neue und praktische Anwendungsfelder erschließen lassen. Die Untersuchung basiert auf einer Software–Simulation und einer theoretischen Analyse des Verfahrens, indem Konvergenzanalyse, Vorschläge zur Parameterwahl und Skalierbarkeit des Verfahrens als Schwerpunkte behandelt werden.

Die richtigen Netzparameter (Lernparameter, Anzahl der Neuronen in der verborgenen Schicht, u.s.w.) zu wählen ist für das BP–Netz immer eine Herausforderung, weil dies die Konvergenzgeschwindigkeit oder sogar die Konvergenz des Trainings überhaupt entscheiden kann. Bei der Software–Implementierung des BP–Netzes kann man während des Trainings verschiedene Parameter–Kombinationen ausprobieren und dann die besten wählen. Dagegen ist dies bei der Hardware–Implementierung nicht so leicht zu erreichen. Eine umfangreiche, häufige und damit flexible Einstellung der Netzparameter ist bei der Hardware–Lösung nicht vorgesehen. Man kann höchstens einige Parameter während des Trainings oder vorher in einem kleinen Umfang stufenweise einstellen. Wegen der hohen Geschwindigkeit gegenüber der Software–Implementierung des BP–Netzes sind die feinen Einstellungen der Trainingsparameter bei der Hardware nicht entscheidend, solange das Training konvergiert. In dem Sinne wäre es ausreichend, daß die oberen und unteren Grenzwerte der Parameter gefunden werden, so daß die Konvergenz des Trainings immer gewährleistet wird, wenn die Trainingsparameter dazwischen liegen. In der folgenden Arbeit werden zuerst Anstrengungen unternommen, um solche oberen und unteren Grenzen der Trainingsparameter herauszufinden und einen Leitfaden zur Parameterwahl für das Training zu erstellen. Darüber hinaus wird die Empfindlichkeit des Trainings gegenüber den Werten der gewählten Trainingsparameter unter verschiedenen Hardware–Komponenten analysiert und untersucht. Das Ziel besteht darin, daß nur solche Hardware–Komponenten verwendet werden sollen, mit denen die Konvergenz des Netzes weniger empfindlich auf die Trainingsparameter reagiert. Mit anderen Worten: es soll mehr Toleranz für die Werte der gewählten Trainingsparameter erreicht werden.

Die mögliche Skalierbarkeit des in [51] vorgeschlagenen Verfahrens soll durch reine Software–Simulation anhand umfangreicher Beispiele untersucht und bestätigt werden. Dafür muß zunächst geklärt werden, welche Bauteile des vorgestellten Verfahrens eine solche Skalierung möglicherweise behindern. Wenn solche Einschränkungen wirklich bestehen, dann ist weiterhin nach Techniken zu suchen, die als Maßnahmen gegen solche Einschränkungen eingesetzt werden können.

1.3 Aufbau der Arbeit

Das Kapitel 1 sollte eine kurze Einleitung zur Problemstellung sowie Zielsetzung der vorliegenden Arbeit geben. Über die Hintergründe dieser Arbeit und den aktuellen Forschungsstand auf dem Gebiet der Hardware–Implementierung des ANN wurde ebenfalls gesprochen.

Die Grundlagen des ANN in Kapitel 2 dienen als Übergang zur weiteren detaillierten Diskussion in den nachstehenden Kapiteln. Außer den grundlegenden Begriffen von ANN und BP–Netz wird die Umsetzung von konventionellen zu stochastischen Rechentechniken bei der Implementierung des BP–Netzes als Schwerpunkt erklärt. Dies betrifft vor allem die sogenannte Codierung, die eine Maschinenvariable (eine physikalische Größe) in eine binäre Bitfolge umwandelt. Die Implementierung der Elementaroperationen (Addition, Multiplikation, usw.) mit stochastischen Rechenwerken und die Implementierung der nichtlinearen Überführungsfunktion und deren Ableitung werden hier kurz zusammengefaßt.

Im Kapitel 3 werden die Strategien zur Parameterwahl für das vorgestellte Verfahren durch Herstellung eines mathematischen Zusammenhangs zwischen dem Verfahren und dem konventionellen BP–Algorithmus herausgefunden und festgelegt. Durch das Studieren der speichernden Glieder (INDIE und ADDIE), die sowohl für die Speicherungen als auch Modifikationen der Gewichte zuständig sind, wird eine 1:1–Abbildung der Lernparameter im vorgestellten Verfahren zu denen des konventionellen BP–Algorithmus mathematisch abgeleitet. Mit Hilfe dieser 1:1–Abbildung kann ein Leitfaden zur Auswahl der Lernparameter für das Training des Netzes festgestellt werden, weil die Bedeutung und Einflußstärke der Lernparameter im konventionellen BP–Algorithmus wie Lernrate γ, Steilheit β und Momentum–Konstante α aus der Literatur bekannt sind und viele Vorschläge zu ihrer Wahl gemacht wurden. Darüber hinaus werden Unter– und Obergrenze der Lernparameter des vorgestellten Verfahrens festgestellt und durch einige Beispiele bestätigt.

Kapitel 4 ist das zentrale Kapitel dieser Arbeit. Es untersucht die potentiellen Einschränkungen im vorgestellten Verfahren im Hinblick auf ihre Auswirkungen auf die Konvergenz beim Training. Die Einschränkungen ergeben sich aus der Verwendung der stochastischen Rechentechnik. Die Untersuchung wird sowohl durch mathematische Herleitung als auch durch Software–Simulation mit geeigneten Beispielen durchgeführt. Über die Zusammenwirkung aller Einschränkungen auf die Konvergenzeigenschaften des Verfahrens wird ebenfalls gesprochen. Danach werden mögliche Maßnahmen gegen die negativen Auswirkungen der Einschränkungen vorgestellt. Den Schwerpunkt bildet die Einführung eines modifizierten Neurons, das

zum Mechanismus des vorhandenen Verfahrens paßt und viele positive Eigenschaften mit sich bringt. Um die Auswirkungen der Gegenmaßnahmen zu bestätigen, werden geeignete Beispielklassen ausgewählt, die unter vertretbarem Rechenaufwand per Software–Simulation Trainingsergebnisse liefern können. Hinsichtlich der Ergebnisse aus der Software–Simulation schließt das Kapitel mit einer bewertenden Diskussion der Gegenmaßnahmen und einer Schlußbemerkung zum vorgestellten Verfahren ab.

Im letzten Kapitel 5 werden alle Untersuchungen dieser Arbeit in kurzer Form zusammengefaßt und ein Ausblick auf bisher noch offen gebliebene wissenschaftliche Fragestellungen gegeben. Eine quantitative Aussage über die Skalierbarkeit des vorgestellten Verfahrens wird in dieser Arbeit nicht getroffen; stattdessen wird die Verwendung des Gesetzes der großen Zahl aus der Theorie der Wahrscheinlichkeit und Statistik vorgeschlagen, die zu einer möglichen quantitativen Aussage der Skalierbarkeit führen könnte. Die Grundlagen des Gesetzes der großen Zahl und ihr Zusammenhang zum Thema der Skalierbarkeit des vorgestellten Verfahrens werden im Anhang B erläutert.

2 Einführende Grundlagen zu Backpropagation–Netzen

Backpropagation–Netze (BP–Netze als Abkürzung) sind mehrschichtige vorwärtsgerichtete Netze mit einer Sigmoidfunktion als Aktivierungsfunktion und Backpropagation als Lernverfahren. Die Begriffe in dieser Definition werden nachfolgend detailliert erläutert.

2.1 Netzaufbau

Ein neuronales Netz besteht aus Neuronen und Verknüpfungen. Neuronen sind Verarbeitungseinheiten von Signalen der Außenwelt oder anderer Neuronen. In der vorliegenden Arbeit wird nur von Neuronen mit Sigmoid- oder sigmoid–ähnlichen Funktionen als Aktivierungsfunktion gesprochen. Verknüpfungen sind Verbindungen eines Neurons zu anderen Neuronen oder der physikalischen Außenwelt. Die Stärke der Verbindung zwischen einem Neuron i und einem Neuron j wird durch das sogenannte Gewicht repräsentiert, welches der Synapse in einem natürlichen Nervensystem entspricht und in der Kurzform als w_{ij} bezeichnet wird. Die Abbildung 2.1 zeigt zwei Neuronen und ihre Verbindung.

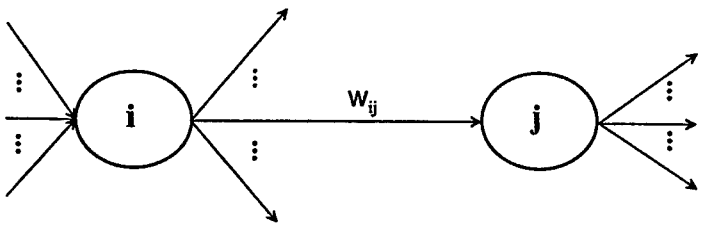


Abb. 2.1: *Neuronen und Verknüpfungen*

Verschiedene Verbindungen zwischen Neuronen führen zu unterschiedlichen Topologien neuronaler Netze. Man kann sie aber in zwei wesentliche Klassen unterteilen, die sogenannten vorwärtsgerichteten Netze (*feedforward networks*) sowie Netze mit Rückkopplungen (*feedback networks*). Offensichtlich enthält ein vorwärtsgerichtetes Netz keine Rückkopplung und die Informationen für die Verarbeitung fließen nur in eine festgelegte Richtung. Diese Arbeit beschränkt sich auf die Betrachtung vorwärtsgerichteter Netze.

2.1.1 Multilayer Netz (MLN)

Eine häufig anzutreffende Architektur für vorwärtsgerichtete Netze ergibt sich durch die Hintereinanderschaltung mehrerer Neuronenschichten, wobei die Neuronen einer Schicht nicht miteinander oder mit sich selbst verbunden sind. Verbindungen bestehen ausschließlich von jedem Neuron einer Schicht zu den Neuronen der nachstehenden Schicht. Die Schicht, von der das Netz eine Eingabe von der Außenwelt erhält, wird als Eingangsschicht bezeichnet. Hier findet keine Informationsverarbeitung statt, sondern es wird nur eine Verbindung zur Außenwelt hergestellt. Somit wird sie für die Anzahl der Schichten eines Netzes nicht gezählt. Die Schicht, von der die Ausgabe eines Netzes erfolgt, wird als Ausgangsschicht bezeichnet. Eine Zwischenschicht, welche weder direkte Eingaben bekommt noch direkte Ausgaben liefern kann, nennt man verborgene Schicht (*hidden layer*). Neuronale Netze mit einer solchen schichtweisen Architektur bezeichnet man als mehrschichtige vorwärtsgerichtete Netze (*Multilayer Feedforward Networks*, MLN).
Aus mathematischer Sicht ist ein NN eine Abbildung, die gewisse Wertemengen (Eingaben des Netzes) in andere, zugeordnete Wertemengen (Ausgaben) abbilden kann.

$$\Phi : \vec{x} \mapsto \vec{y} \qquad \vec{x} \in R^n, \vec{y} \in R^m \tag{2.1}$$

In der Literatur ([10], [21], [37], [42] und [50]) wurde nachgewiesen, daß sich mit Hilfe eines NN mit nur einer verborgenen Schicht theoretisch jede reelle Funktion beliebig genau annähern läßt. Daher beschränkt man sich bei praktischen Problemstellungen meist auf zweischichtige Netze, die in der Regel weniger Implementierungsaufwand als Netze mit mehreren verborgenen Schichten verursachen. Im Einzelfall muß aber untersucht werden, mit welcher Anzahl von Schichten und Neuronen je Schicht eine Problemstellung am besten gelöst werden kann.

2.1.2 Anzahl der Neuronen in einem MLN

2.1.2.1 Ein– und Ausgangsschichten

Wie oben gesagt, bilden Ein– und Ausgangsschichten eines MLN die Schnittstelle zur Außenwelt. Dadurch sind ihre Dimensionen schon von der zu lösenden Aufgabenstellung bestimmt. Wenn z. B. die Abbildung in der Gleichung (2.1) durch ein MLN implementiert werden soll, dann besitzen dessen Ein– und Ausgangsschichten jeweils n und m Neuronen. Oft ist es jedoch sinnvoll, die ursprüngliche Aufgabenstellung umzuformen, damit die Aufgabe leichter von einem MLN bewältigt werden kann. Beispiel:

Aufgabenstellung: Erkennung von hexadezimalen Zahlen im Bereich 0...F durch ein MLN

Eingabe: n Pixel eines Binärwertbildes von hexadezimalen Zahlen

Ausgabe: hexadezimale Zahl als Wert

Beim Lösen dieser Aufgabenstellung benötigt das Netz offensichtlich n Eingänge, wobei sich n je nach Auflösung des Binärwertbildes ändern kann. Für die Ausgabe reichen vier Neuronen gut aus, weil eine Binärstelle der hexadezimalen Zahl einem Neuron entsprechen kann. In der Praxis ist es jedoch besser, jeder Klasse, d.h. jeder einzelnen Hexadezimalzahl im vorgegebenen Bereich, ein Neuron zuzuordnen, um das Training des Netzes zu erleichtern ([28] und [30]). Dies führt zu 16 Neuronen in der Ausgangsschicht.

2.1.2.2 Verborgene Schichten

Über die Festlegung der Anzahl von Neuronen in verborgenen Schichten findet sich vergleichsweise wenig in der Literatur. Diese Anzahl bestimmt die Netzkapazität und ist von der Aufgabenstellung abhängig. Die Frage, wieviele Neuronen man für die verborgene Schicht verwenden sollte, ähnelt der, von welchem Grad ein Polynom sein sollte, um eine gegebene Funktion am besten anzunähern. Rojas [55] hat eine untere Schranke für die Anzahl der Neuronen in der verborgenen Schicht aufgewiesen, Zitat: „Eine verborgene Schicht muß so viele Neuronen haben, wie die Trainingsmenge Vektoren hat". Dies mag eine hinreichende Bedingung sein, ist aber zu hart für die Wirklichkeit. In der Tat kann ein Netz in vielen Fällen mit weniger Neuronen eine gegebene Aufgabe erfolgreich lösen (siehe [37]). Werden zu viele Neuronen für die verborgene Schicht ausgewählt, kann sich einerseits der Aufwand für das Training des Netzes erheblich erhöhen, andererseits kann es übertrainiert werden (*overfitting*). Damit nimmt die Generalisierungsfähigkeit bezüglich untrainierter Muster ab. Deshalb muß man versuchen, so wenige Neuronen wie möglich der verborgenen Schicht zuzuordnen (vorausgesetzt, daß das Training des Netzes konvergiert), um *overfitting* zu vermeiden. In [28], [37] und [72] werden einige gute praktische Hinweise über die Auswahl der Neuronenanzahl in der verborgenen Schicht gegeben.

2.1.3 Verbindung zwischen Schichten

Ein anderer und entscheidender Faktor für Netztopologien ist die Art der Verbindungen zwischen den Schichten. Sie hat einen wichtigen Einfluß auf die Netzdynamik. Dies kann sowohl die Netzkapazität als auch die Trainingsgeschwindigkeit stark beeinflussen, insbesondere dann, wenn das NN

zur Klassifikation verwendet wird. Es ist in den Klassifikationsanwendungen allgemein üblich, daß jeder Klasse ein Ausgangsneuron zugeordnet wird.

2.1.3.1 Vollverbindung

Unter Vollverbindung soll hier verstanden werden, daß jedes Neuron einer Schicht zu jedem Neuron der nachfolgenden Schicht (falls es eine gibt) eine Verbindung hat. Dies ist das sogenannte ACON (*All–Class–in–One–Network*). Abbildung 2.2 zeigt ein Beispiel.

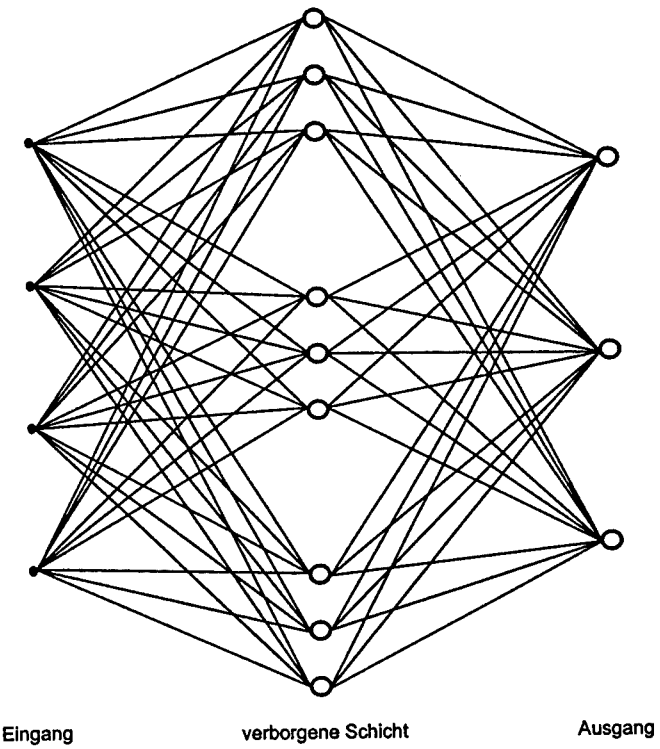


Abb. 2.2: *Ein 4–9–3–Netz mit ACON–Struktur*

Wenn die Neuronenanzahl in der Ausgangsschicht groß ist, kann das gesamte Netz damit auch sehr groß werden. Es wurde in der Literatur [30] darauf hingewiesen, daß die Konvergenzrate eines ACON bei der Erhöhung der Neuronenanzahl des Netzes drastisch sinkt. So ist das ACON nur für Anwendungen geeignet, die wenige Neuronen in der Ausgangsschicht erfordern.

2.1.3.2 Teilverbindung

Wenn die von den Anwendungen bestimmte Neuronenanzahl in der Ausgangsschicht sehr groß ist, dann ist das sogenannte OCON (*One–Class–in–One–Network*) eine geeignete Struktur. In diesem Fall sind die Neuronen der verborgenen Schicht und der Ausgangsschicht nur teilweise miteinander verbunden und das ganze Netz wird entsprechend in mehrere Teilnetze partitioniert. Jedes Teilnetz besitzt dabei nur wenige Neuronen (typischerweise 3–5 Neuronen) in seiner verborgenen Schicht. Abbildung 2.3 zeigt dazu ein Beispiel.

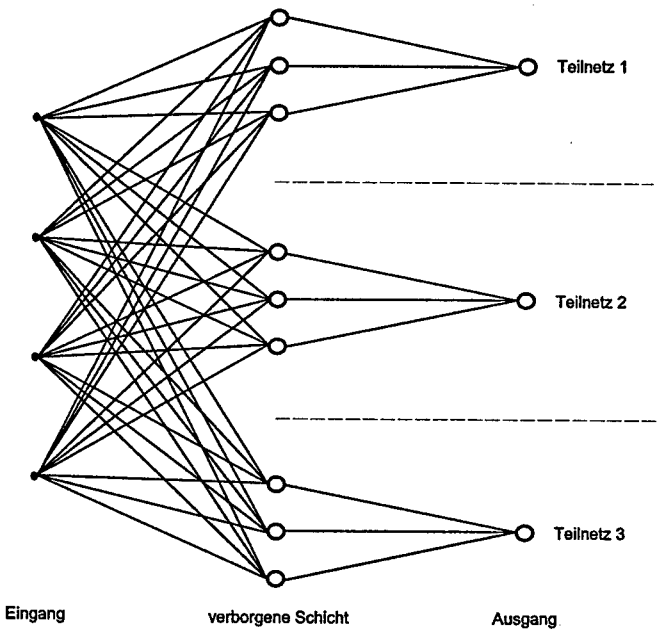


Abb. 2.3: Eine OCON–Struktur je Teilnetz mit drei Neuronen in der verborgenen Schicht

In dieser Struktur ist jedes Teilnetz nur für einen Ausgang (entsprechend einer Klasse) zuständig. So läßt sich die Gewichtebelegung des Netzes relativ leicht durch das Training erreichen. Folglich kann man sich viel Zeit in der Trainingsphase ersparen und das Netz kann damit eine bessere Leistung in der Arbeitsphase anbieten. Außerdem könnte diese Struktur eine mögliche Maßnahme gegen die negative Auswirkung des sogenannten $\frac{1}{N}$–Verfahrens bei Netzen in stochastischer Rechentechnik darstellen (siehe [51]).

2.2 Lernalgorithmus

Neben der Netztopologie ist das Lernverfahren selbst eine wichtige Komponente neuronaler Netze bzw. Verfahren. Das Lernverfahren erlaubt es, daß ein Netz eine gegebene Aufgabe selbständig aus Beispielen erlernt. Allgemein unterscheidet man zwei Arten von Lernen in neuronalen Netzen: **unüberwachtes Lernen** (*unsupervised learning*) und **überwachtes Lernen** (*supervised learning*). In dieser Arbeit wird nur auf überwachtes Lernen eingegangen.

Das Ziel eines Lernverfahrens ist es, die Gewichte des Netzes so zu belegen, daß das Netz nach wiederholter Präsentation der Paare von Eingangs– und Ausgangsmustern dieselbe Assoziation sowohl für Musterpaare in der Trainingsmenge als auch für unbekannte, ähnliche Eingaben (Generalisierung) vornehmen kann. Beim überwachten Lernen wird das Ziel mit Hilfe eines externen „*Lehrers*" erreicht, der dem Netz Eingangsmuster aus der Trainingsmenge und zugehörige Ausgangsmuster präsentiert. Dies bedeutet, daß dem Netz immer gleichzeitig ein vollständig spezifiziertes Eingangsmuster und ein korrektes bzw. optimales und korrespondierendes Ausgangsmuster vorliegt. Neben dem „*Lehrer*" existiert auch eine Lernregel; sie entscheidet, wie die Gewichte in Bezug auf „Soll" und „Ist" an den Netzausgängen iterativ modifiziert werden sollen, um eine optimale Belegung der Gewichte zu verwirklichen. Viele Lernregeln wurden von verschiedenen Wissenschaftlern unabhängig voneinander und für unterschiedliche Anwendungen entwickelt und veröffentlicht. Backpropagation (*generalized delta rule*) ist die bekannteste und populärste Lernregel für mehrschichtige vorwärtsgerichtete Netze.

2.2.1 Warum Backpropagation?

Der Algorithmus „Backpropagation" wurde ursprünglich von Bryson und Ho [1] 1969 für optimale Kontrolle entwickelt und 1974 von Werbos [71] als eine Verallgemeinerung von statistischen Regressionsmethoden wieder entdeckt. Aber erst die Arbeit von Rumelhart, Hinton und Williams [57] machte diesen Algorithmus bekannt und populär. Ihr wesentlicher Beitrag war es, daß die Lernfehler für Neuronen in den verborgenen Schichten (*internal representations*) durch zurückgeführte Fehlermaße aus der nachfolgenden Schicht berechnet werden können.

Der BP–Algorithmus ist leicht zu verstehen und zu implementieren. Die Lokalität von BP, daß nämlich nur lokale Informationen in die Kalkulation einbezogen werden, ist eine wichtige Eigenschaft für seinen Einsatz zum Training neuronaler Netze. So wird die Parallelität der Arbeitsweise von NNs gewährleistet.

Der BP–Algorithmus ist eine Erweiterung des LMS (*least mean square*)–Verfahrens, dessen Prinzip und Eigenschaften in der mathematischen Literatur bereits umfassend behandelt wurden. Per Iteration wird das Minimum der Fehlerfunktion eines bestimmten Lernproblems durch Abstieg in Gradientenrichtung gesucht. Dies setzt lediglich voraus, daß der Gradient der Fehlerfunktion für alle Punkte des Gewichteraums existieren muß, d.h. die partiellen Ableitungen der Fehlerfunktion nach den einzelnen Gewichten müssen überall definiert sein.
Wegen seiner Einfachheit, der guten Anpassung an die parallele Arbeitsweise von NNs und wegen der fundierten mathematischen Grundlage hat sich der BP–Algorithmus bei seinem Einsatz in unterschiedlichen NN–Anwendungen recht erfolgreich durchsetzen können. Aus diesen Gründen bildet er die Grundlage des vorliegenden stochastischen Verfahrens.

2.2.2 Lernen mit Backpropagation (BP)

2.2.2.1 Problemstellung

Vorgegeben ist eine Trainingsmenge Γ, die aus P Vektorpaaren von n–dimensionalen Eingabevektoren und m–dimensionalen Ausgabevektoren besteht, d.h.

$$\Gamma = \{(\vec{u}_i, \vec{t}_i) | \vec{u}_i \in R^n, \vec{t}_i \in R^m; i = 1, 2.., P\} \tag{2.2}$$

Die Aufgabe des Trainings ist es, das Netz durch BP lernen zu lassen, jeden Vektor $\vec{u}_i$ in den entsprechenden $\vec{t}_i$ so genau wie möglich abzubilden. Um den Lernfortschritt und damit den Näherungsgrad des Netzes an die Soll–Abbildung quantitativ beschreiben zu können, ist ein Maß erforderlich. Dies ist der sogenannte Lernfehler E, der als eine Funktion von den Gewichten des Netzes betrachtet werden soll:

$$E = \sum_{i=1}^{P} \|\vec{t}_i - \vec{y}_i\|^2 = \frac{1}{2} \sum_{i=1}^{P} \sum_{j=1}^{m} (t_{ij} - y_{ij})^2 \tag{2.3}$$

Die $\vec{y}_i$ sind die Ist–Ausgaben des Netzes während des Trainingsverlaufs. Die t_{ij} und y_{ij} sind j–te Komponenten der entsprechenden Vektoren $\vec{t}_i$ und $\vec{y}_i$. Offensichtlich ist der Lernfehler E in der Gleichung (2.3) von der Dimension des Netzausgangs und der Anzahl der Trainingsmuster abhängig. Darum ist es zweckmäßig, einen durchschnittlichen Lernfehler zu verwenden. Er lautet nach [37]:

$$\bar{E} = \frac{1}{P} \sum_{i=1}^{P} \frac{1}{m} \|\vec{t}_i - \vec{y}_i\|^2 = \frac{1}{Pm} \sum_{i=1}^{P} \sum_{j=1}^{m} (t_{ij} - y_{ij})^2 \tag{2.4}$$

Die Gleichung (2.4) ist Grundlage der weiteren Betrachtungen.

2.2.2.2 Arithmetische Operationen

Der BP–Algorithmus kann in zwei Phasen unterteilt werden: Propagation– und Backpropagation–Phase.

Sei:

N	Anzahl der Schichten des Netzes
n_l	Anzahl der Neuronen in der Schicht l, $l = 1, ..., N$
$\vec{x}^{[l]}$	Aktivierungsvektor der Schicht l $\vec{x}^{[0]}$ = Eingangsvektor $\vec{u}$ des Netzes
$x_j^{[l]}$	j–te Komponente von Vektor $\vec{x}^{[l]}$
$\vec{a}^{[l]}$	gewichteter Eingangsvektor der Schicht l
$a_j^{[l]}$	j–te Komponente von Vektor $\vec{a}^{[l]}$
$\vec{t}$	Soll–Vektor aus der Trainingsmenge
t_j	j–te Komponente von Vektor $\vec{t}$
$s(.)$	Sigmoidfunktion, $s(x) = \frac{1}{1+e^{-\mu x}}$
μ	Steilheit der Sigmoidfunktion
$s'(.)$	Ableitung der Sigmoidfunktion, $s'(x) = \mu s(x)[1 - s(x)]$
$w_{ij}^{[l]}(k)$	von der k-ten Iteration gewonnenes Gewicht an der Verbindung zwischen Neuron i in der Schicht $l-1$ und Neuron j in der Schicht l
$\Delta w_{ij}^{[l]}(k)$	Gewichtsänderung für $w_{ij}^{[l]}(k)$ bei der k-ten Iteration.

Dann läßt sich für jedes feste Vektorpaar $(\vec{u}, \vec{t})$ aus der Trainingsmenge der ursprüngliche BP–Algorithmus durch folgende Gleichungen beschreiben:

- Propagation

$$x_j^{[l]} = s(a_j^{[l]}) = s(\sum_{i=0}^{n_{l-1}} w_{ij}^{[l]} x_i^{[l-1]}) \quad l = 1, ..., N; \quad j = 1, ..., n_l \tag{2.5}$$

- Backpropagation

$$w_{ij}^{[l]}(k+1) = w_{ij}^{[l]}(k) + \Delta w_{ij}^{[l]}(k+1) \tag{2.6}$$

$$\Delta w_{ij}^{[l]}(k+1) = \gamma \delta_j^{[l]} x_i^{[l-1]} \quad j = 1, ..., n_l \quad i = 1, ..., n_{l-1} \tag{2.7}$$

$$\delta_j^{[l]} = s'(a_j^{[l]}) \sum_{i=0}^{n_{l+1}} \delta_i^{[l+1]} w_{ij}^{[l+1]}(k) \quad j = 1, ..., n_l \quad l = N-1, ..., 1 \tag{2.8}$$

$$\delta_j^{[N]} = s'(a_j^{[N]})(t_j - x_j^{[N]}) \qquad j = 1, ..., n_N \tag{2.9}$$

Der Eingangsvektor jedes Neurons wurde schon um eine Dimension erweitert, um einen Bias–Eingang einzubeziehen. So entspricht jeder Index 0 in den Summentermen der obigen Gleichungen der Bias–Leitung.

Das γ in der Gleichung (2.7) wird Lernrate genannt, weil seine Größe die Konvergenz und die Lerngeschwindigkeit des BP–Algorithmus bestimmt.

Ein Schichtendurchlauf (z.B. für Schicht l) des BP erfordert einen Rechenaufwand von $O(n_l^2)$ Multiplikationen und Additionen zweier Werte, darüber hinaus noch $O(n_l)$ nichtlineare Überführungen, repräsentiert durch Sigmoidfunktionen und ihre Ableitungen. Die Subtraktion in Gleichung (2.9) wird hier im Hinblick auf die stochastische Rechentechnik als Komplement einer Addition betrachtet.

2.2.2.3 Ablauf des Trainings

Die klare Fassung des Trainingsablaufs durch geeignete Begriffe ist zur Formulierung einer schematischen Vorschrift für den Betrieb eines Netzes in einem umgebenden System wesentlich. Aus diesem Grund sollen zuerst einige Begriffe definiert werden:

- Eine **Präsentation** eines Musterpaares bedeutet, daß ein Eingabevektor aus der Trainingsmenge in den Eingang des Netzes eingespeist und mit der Gleichung (2.5) schichtenweise berechnet wird, bis eine Netzausgabe (als Ist–Wert gekennzeichnet) beim Netzausgang auftaucht. Nach dem Vergleich zwischen Soll– und Ist–Wert wird der entsprechende Fehler in den Ausgang des Netzes eingeführt und die Gleichungen (2.7) bis (2.9) werden schichtenweise rückwärts bis zum Eingang des Netzes durchgeführt.

- Eine **Epoche** beinhaltet die Präsentationen aller Musterpaare aus der Trainingsmenge oder ihrer Teilmenge.

- ***Online–Training*** heißt, daß die Gewichtskorrekturen nach jeder Präsentation eines Musterpaares aus der Trainingsmenge durchgeführt werden (Gleichung (2.6)).

- **Batch–Verfahren** bedeutet, daß die Gewichtskorrekturen (Gleichung (2.6)) nur nach jeder Epoche berechnet werden. In diesem Fall ist $\Delta w_{ij}^{[l]}(k+1)$ in der Gleichung eine Akkumulation der Gewichtsänderungen $\Delta w_{ij}^{[l]}(k+1,r)$ (r ist der Index für Musterpaare in der Trainingsmenge) für jede Präsentation in der Epoche, d.h.:

$$\Delta w_{ij}^{[l]}(k+1) = \sum_{r=1}^{P} \Delta w_{ij}^{[l]}(k+1,r) \tag{2.10}$$

Am Anfang müssen alle Gewichte zufällig oder mit gewissen unterschiedlichen Werten initialisiert werden. Die Anfangswerte haben Einfluß auf die Konvergenz und die Lerngeschwindigkeit von BP. Danach wird dem Netz wiederholt solange die Trainingsmenge präsentiert, bis eine Stop–Bedingung erfüllt wird. Die Stop–Bedingung wird nach jeder Epoche getestet. Eine häufig verwendete Bedingung ist die Prüfung, ob der aktuelle Fehler $\bar{E}$ kleiner als eine vorgegebene Konstante ε ist oder ob die Anzahl der schon durchgeführten Epochen einen bestimmten Wert (auch empirisch vorgegeben) überschritten hat.

Die Gewichtskorrekturen (Gleichung (2.6)) können gemäß Batch–Verfahren oder *Online–Training* durchgeführt werden. Beim Batch–Verfahren wird die echte Gradientenrichtung der Gesamtfehlerfunktion für die Gewichtskorrekturen verwendet. Der Rechenaufwand ist jedoch größer als beim *Online–Training*, bei dem die für die Korrekturen verwendete Richtung im Gewichteraum aber nicht mit der Gradientenrichtung übereinstimmt. Beim *Online–Training* sollte das Musterpaar zufällig aus der Trainingsmenge ausgewählt werden, damit die Korrekturrichtung um den Gradienten oszilliert und der maximal absteigenden Richtung auf der Fehlerfläche im Durchschnitt gefolgt wird [55].

2.2.3 Varianten des BP

Ein wesentlicher Nachteil des BP ist es, daß das Lernen in vielen Fällen zu langsam abläuft und/oder zu einem lokalen Minimum konvergiert. Die flachen Täler, die durch die Verwendung der Sigmoide entstehen, führen insbesondere bei kleinen Schrittgrößen zu einer großen Anzahl von Iterationen des Algorithmus. Um die Konvergenzgeschwindigkeit des BP zu erhöhen, sind eine Reihe von Varianten des BP vorgeschlagen worden, die in zwei Gruppen klassifiziert werden können. In der ersten Gruppe sind Varianten, welche die Delta–Regel (Gleichung (2.7)) beibehalten. Die entsprechenden Methoden sind in [2], [9], [20], [58] und [60] detailliert beschrieben. Zu der

zweiten Gruppe gehören diejenigen, die statt Gleichung (2.7) eine andere Regel nutzen, um die Gewichte zu korrigieren. Ein Beispiel ist *Quickprop*, das eine Methode zweiter Ordnung darstellt [9]. Die einfachste und populärste Technik ist es, einen Impulsterm in die Gleichung (2.7) einzuführen. Dies ist das sogenannte Backpropagation mit Impulsterm. Hierbei ergibt sich die Gleichung für die Korrektur der Gewichte zu:

$$\Delta w_{ij}^{[l]}(k+1) = \gamma \delta_j^{[l]} x_i^{[l-1]} + \alpha \Delta w_{ij}^{[l]}(k) \quad j = 1, n_l \quad i = 1, ..., n_{l-1} \qquad (2.11)$$

Dabei wird α, das empirisch bestimmt wird, als **Momentum–Konstante** bezeichnet. Wegen seiner Einfachheit und Effizienz wurde Backpropagation mit Impulsterm als Lern–Algorithmus zum Ausgangspunkt des nachfolgend beschriebenen stochastischen Verfahrens gewählt.

2.3 Umsetzung in stochastische Rechentechnik

Im untersuchten stochastischen Verfahren wird eine Art von Wertedarstellung verwendet, in der Bitströme aus binären Pulsen die Informationsträger sind. Daher sollen die Codierung, die arithmetischen Operationen und die Implementierung der Nichtlinearität auf der Bitstrom–Ebene erläutert werden, wozu einige Begriffe aus [51] herangezogen werden müssen:

Eine **Problemvariable** hat einen problemabhängigen endlichen Wertebereich sowie eine theoretisch ideale Genauigkeit.

Eine **Maschinenvariable** M kann durch Quantisierung und lineare Transformation aus einer Problemvariablen gewonnen werden. Sie liegt ausschließlich im Intervall $[-1, 1]$ und hat eine bestimmte Anzahl binärer Stellen.

Ein **Bitstrom** B ist eine zufällige und jeweils zu allen anderen Bitströmen des Netzes stochastisch unabhängige Folge von Nullen und Einsen auf einer Leitung. Dabei bezeichnet $\bar{B}$ die Negation der Folge B. Demzufolge gilt die Wahrscheinlichkeitsberechnung: $P(B = 1) + P(\bar{B} = 1) = 1$.

Die Problemvariablen existieren nur außerhalb des Netzes. Sie werden vor den Netzeingängen und den Sollwerteingängen zu Maschinenvariablen transformiert bzw. normiert. Innerhalb des Netzes wird nur der Begriff Maschinenvariable verwendet, d.h. der größtmögliche Betrag eines Wertes ist

Eins und alle arithmetischen Operationen sowie Übertragungs– und Speicherfunktion haben den Definitionsbereich der Maschinenvariablen $[-1, 1]$. Dieser Wertebereich wird in den folgenden Kapiteln häufig als **M–Bereich** bezeichnet. Hingegen wird das Intervall $[0, 1]$ **W–Bereich** genannt, welcher den Wertebereich der Wahrscheinlichkeit repräsentiert.

2.3.1 Codierung und Decodierung

Unter Codierung versteht man hier die Umsetzung einer Maschinenvariablen in eine binäre Bitfolge, deren Wahrscheinlichkeit für das Erscheinen einer Eins nach der folgenden Vorschrift dem Wert der Maschinenvariablen entspricht:

$$P(B = 1) = \frac{M + 1}{2} \tag{2.12}$$

Diese Abbildung wurde von Massen [36] als bipolare 1–Leiter–Darstellung für vorzeichenbehaftete Variablen bezeichnet. Wird die Nomenklatur von [51] übernommen, bei der die Wahrscheinlichkeit des Auftretens einer Eins in einem Bitstrom B mit Großbuchstaben und die damit verbundene Maschinenvariable mit dem gleichen, aber kleinen Buchstaben bezeichnet wird, so erhält man:

$$X = P(B_x = 1) = \frac{x + 1}{2} \tag{2.13}$$

$$x = 2 * X - 1 = 2 * P(B_x = 1) - 1 \tag{2.14}$$

Die Schaltung zur Schaffung der Übergänge von den deterministischen Signaldarstellungen in die entsprechenden statistischen soll als stochastischer Codierer bezeichnet werden. Weil es sich in der vorliegenden Arbeit nur um die digitale Darstellung des Signals handelt, wird hier ein sogenannter **DSC**, Digital–Stochastik–Codierer [36], für den Codierungsvorgang konstruiert. Die Einzelheiten für den Aufbau eines DSC werden im folgenden Abschnitt besprochen.

2.3.1.1 Implementierung

In der Literatur wurden unterschiedliche Schaltungen für die Hardware–Implementierung der obigen Codierung vorgeschlagen. Ganz generell kann sie durch Vergleich der Maschinenvariablen mit einer genauso langen Zufallszahl realisiert werden. Diese Codierung zeigt Abbildung 2.4. Die notwendige Zufallszahl für die Implementierung muß genauso viele Stellen aufweisen wie die Maschinenvariable.

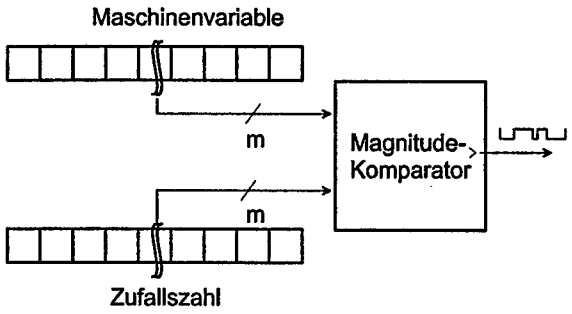


Abb. 2.4: Codierung mit komparatorischer Schaltung

In dieser Arbeit wird eine Schaltung verwendet, die als sequentielle Codierung bezeichnet wird und für die Codierung einer m Bit breiten Maschinenvariablen nur eine einzige Zufallsfolge benötigt [51]. Abbildung 2.5 zeigt die Struktur dieser Schaltung.

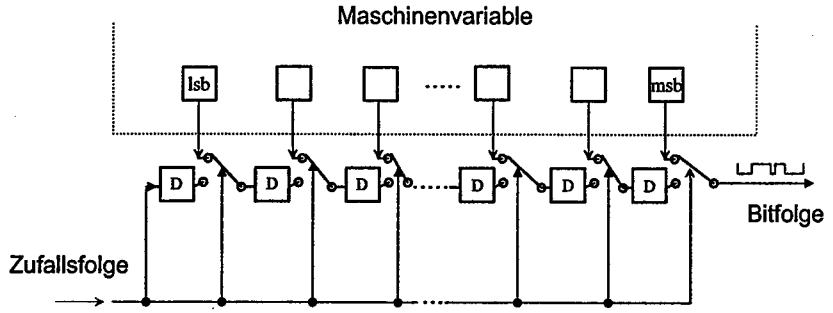


Abb. 2.5: Codierung mit nur einer binären Zufallsfolge durch Speichern der selektierten Bits der Maschinenvariablen (nach [51])

Hier wird ausgenutzt, daß die Verbundwahrscheinlichkeiten einer idealen Zufallsquelle der binären Stellengewichtung einer sehr langen Zahl entsprechen. Dies erfordert, daß die codierte Maschinenvariable während der Codierung stationär sein muß.

2.3.1.2 Stochastische Streuung

Die Wiedergewinnung der Maschinenvariablen aus der entsprechenden Bitfolge wird als Decodierung bezeichnet. Theoretisch kann nur dann die Maschinenvariable exakt wiedergewonnen werden, wenn eine unendlich lan-

ge Stichprobe aus der Bitfolge entnommen werden könnte. In der Praxis kann man jedoch nur aus einer endlich langen Stichprobe einer Bitfolge eine Schätzung für die entsprechende Maschinenvariable vornehmen. Solch eine Schätzung ist um so besser, je größer der Umfang der Stichprobe ist. Dies führt zu einem entsprechenden Schätzungsfehler, der sogenannten stochastischen Streuung. Die blockweise Summierung und gleitende Mittelwertbildung (z.B. im ADDIE in [51]) bilden zwei bekannte Schätzvorgänge [36]. Bei der Summierung für die Decodierung einer Bernoulli–Folge läßt sich die Standardabweichung des aus n Stichproben bestimmten Schätzwertes $\tilde{X}$ mit folgender Gleichung beschreiben:

$$\sigma_{\tilde{X}} = \sqrt{\frac{X(1-X)}{n}} \tag{2.15}$$

Für die gleitende Mittelwertbildung mit einem m–stufigen Zähler ergibt sich die Standardabweichung zu:

$$\sigma_{\tilde{X}} = \sqrt{\frac{X(1-X)}{2^m - 1}} \tag{2.16}$$

Die obigen Gleichungen zeigen, daß für die Randwerte ($X = 1$ oder $X = 0$) keine stochastische Streuung besteht und die maximale Streuung σ_{max} an der Stelle $X = 0.5$ auftaucht, nämlich:

$$\sigma_{max} = \frac{1}{2\sqrt{n}} \tag{2.17}$$

Für das Summierungsverfahren:

$$\sigma_{max} = \frac{1}{2\sqrt{n}} \tag{2.18}$$

Für die gleitende Mittelwertbildung:

$$\sigma_{max} = \frac{1}{2\sqrt{2^m - 1}} \tag{2.19}$$

Der Schätzvorgang benötigt eine bestimmte Zeit. Wenn sich die Eingangsinformation des Decodierers innerhalb dieser Zeit ändert, dann taucht ein sogenannter **Einschwingfehler** auf. Er äußert sich normalerweise in einem zu kleinen Schätzwert $\tilde{X}$.

2.3.2 Arithmetische Operationen

Die wesentliche Motivation, stochastische Rechenwerke zur Implementierung eines BP–Netzes einzusetzen, liegt darin, daß die notwendigen arithmetischen Operationen in BP mit einfachen und damit wenig platzaufwendigen Schaltungen realisiert werden können. Nachfolgend wird die Implementierung von solchen arithmetischen Operationen erläutert. Bei der Diskussion werden die Vereinbarungen für Groß- und Kleinbuchstaben wie weiter oben beschrieben befolgt. Für die folgende Herleitung werden Gleichungen (2.13) und (2.14) mehrfach verwendet.

2.3.2.1 Multiplikation

Die Multiplikation zweier Maschinenvariablen kann durch ein digitaltechnisches Äquivalenzglied sehr einfach erfolgen.
Zu rechnen ist $z = xy$ im Intervall $[-1, 1]$. Das entsprechende Z im Intervall $[0, 1]$ ist:

$$\begin{aligned} Z \stackrel{def}{=} P(B_z = 1) &= \frac{z+1}{2} = \frac{xy+1}{2} = \frac{(2X-1)(2Y-1)+1}{2} \\ &= 2XY - X - Y + 1 \\ &= XY + (1-X)(1-Y) \\ &= XY + \bar{X}\bar{Y} \\ &= P(B_x = 1)P(B_y = 1) + P(\bar{B}_x = 1)P(\bar{B}_y = 1) \end{aligned} \tag{2.20}$$

Wenn B_x und B_y statistisch unabhängig sind, dann ergibt sich daraus:

$$\begin{aligned} Z &= P((B_x \wedge B_y) = 1) + P((\bar{B}_x \wedge \bar{B}_y) = 1) \\ &= P((B_x \wedge B_y) = 1 \cup (\bar{B}_x \wedge \bar{B}_y) = 1) \\ &= P((B_x \wedge B_y \vee \bar{B}_x \wedge \bar{B}_y) = 1) \\ &= P(B_x \equiv B_y) \end{aligned} \tag{2.21}$$

In der Herleitung wurde die Tatsache, daß $(B_x \wedge B_y) = 1$ und $(\bar{B}_x \wedge \bar{B}_y) = 1$ nie gleichzeitig auftreten, implizit verwendet.
Damit ist gezeigt, daß ein Äquivalenzglied zwei stochastisch unabhängige Bitströme —bezogen auf die zugeordneten Maschinenvariablen— multiplikativ verknüpft.

2.3.2.2 Addition

Bei einer Addition zweier oder mehrerer Maschinenvariablen kann das Ergebnis der Addition den Wertebereich der Maschinenvariablen $[-1, 1]$ über-

schreiten. Statt arithmetischer Addition ist deshalb nur eine mittelnde Addition, d.h. Mittelwertbildung, möglich, die zum sogenannten $\frac{1}{N}$–Verfahren führt [51]. Die sich daraus ergebenden Nebenwirkungen werden im folgenden Kapitel ausführlich erläutert.

Zu rechnen ist $z = \frac{1}{2}(x + y)$. Nach dem Abbildungsgesetz der Codierung erhält man:

$$\begin{aligned} Z \equiv P(B_z = 1) &= \frac{z+1}{2} = \frac{[\frac{1}{2}(x+y)+1]}{2} \\ &= \frac{1}{2}(\frac{x+1}{2} + \frac{y+1}{2}) = \frac{1}{2}(X+Y) \\ &= \frac{1}{2}(P(B_x = 1) + P(B_y = 1)) \end{aligned} \tag{2.22}$$

In der Praxis wird eine unabhängige Hilfsfolge B_h mit $P(B_h = 1) = 0.5$ für die Implementierung obiger Gleichung benutzt. Mit dieser Hilfsfolge wird die Entscheidung getroffen, von welcher Folge (B_x oder B_y) ein Bit als das aktuelle Bit der Ergebnisfolge genommen werden soll. In diesem Fall dient die Hilfsfolge als Schalter von B_x und B_y. Die entsprechende Digitalschaltung wurde in [4] als Flatterglied bezeichnet.

Der obige Fall kann leicht zu einer mittelnden Addition beliebig vieler Maschinenvariablen verallgemeinert werden. Auf Grund der Implementierung durch Hilfsfolgen wird die Anzahl der Summanden jedoch auf Potenzen von zwei beschränkt. Dadurch ist die Anzahl der Eingänge jedes Neurons im vorliegenden Verfahren auch auf Potenzen von zwei beschränkt.

2.3.2.3 Subtraktion

Beim BP–Algorithmus wird nur am Netzausgang eine Subtraktion zweier Maschinenvariablen für die Ermittlung des Fehlers zwischen Soll- und Ist–Werten benötigt. Die Subtraktion auf der Bitstrom–Ebene ist ebenfalls durch Mittelung möglich. Die folgende Herleitung zeigt, daß die Subtraktion durch eine mittelnde Addition zwischen dem Minuend und der Negation des Subtrahenden auf der Bitstrom–Ebene implementiert werden kann.

Wenn der Lernfehler definiert wird als

$$E = \frac{1}{2}\sum_{i=1}^{P} \|\vec{t_i} - \vec{y_i}\|^2 = \frac{1}{4}\sum_{i=1}^{P}\sum_{j=1}^{m} (t_{ij} - y_{ij})^2, \tag{2.23}$$

dann ergibt der Unterschied zwischen Gleichungen (2.3) und (2.23) nur einen Faktor $\frac{1}{2}$, der keinen Einfluß auf die Minimumposition im Gewichtsraum

hat. Es kann leicht hergeleitet werden, daß die Gleichung (2.9) im traditionellen BP–Algorithmus zum Folgenden umformuliert werden kann:

$$\delta_j^{[N]} = s'(a_j^{[N]})\frac{1}{2}(t_j - x_j^{[N]}) \qquad j = 1, ..., n_N \tag{2.24}$$

Dadurch ist die Subtraktion am Netzausgang schon richtig normiert. Unsere Aufgabe ist nun, $z = \frac{1}{2}(x-y)$ zu berechnen. Das entsprechende Äquivalent im Intervall $[0,1]$ läßt sich folgendermaßen gewinnen:

$$\begin{aligned} P(B_z = 1) \stackrel{def}{=} Z &= \frac{z+1}{2} = \frac{\frac{1}{2}(x-y)+1}{2} \\ &= \frac{X+1-Y}{2} = \frac{1}{2}(X+\bar{Y}) \\ &= \frac{1}{2}[P(B_x = 1) + P(\bar{B}_y = 1)] \end{aligned} \tag{2.25}$$

Damit ist gezeigt, daß zur Fehlerberechnung am Netzausgang die mittelnde Addition der entsprechenden Bitströme und deren Komplement herangezogen werden darf.

2.3.3 Die nichtlineare Überführungsfunktion

Die nichtlineare Überführungsfunktion und ihre Ableitung sind Kernteile eines Neurons in einem BP–Netz. Das Gradientenabstiegsverfahren verlangt von der Überführungsfunktion nur, daß sie streng monoton steigend ist und sich asymptotisch den Grenzwerten 0 und 1 nähert. Warum die Sigmoidfunktion für den BP–Algorithmus genommen wurde, liegt darin begründet, daß sie sowohl die obigen Bedingungen erfüllt als auch andere gute Eigenschaften (z.B. Einfachheit des Berechnens ihrer Ableitung) besitzt. Jedoch ist ihre exakte Implementierung durch stochastische Rechenwerke kaum möglich. Die Lösung des Problems besteht darin, eine sigmoid–ähnliche Funktion zu implementieren, die die Bedingungen des Gradientenabstiegsverfahrens erfüllt und leicht mit digitalen Schaltungen aufgebaut werden kann. Riemschneider [51] entwickelte eine sogenannte Squashfunktion, die eine S–förmig gekrümmte, monoton steigende Kennlinie aufweist und vor allem digitaltechnisch einfach herzustellen ist. Die Steilheit der Kennlinie kann durch einen Parameter (die sogenannte Runlänge) n eingestellt werden. Diese Squashfunktion läßt sich im Bereich der Wahrscheinlichkeit durch folgende Gleichung beschreiben, die einer gebrochenen rationalen Funktion entspricht:

$$Y(X,n) = \frac{X^n(1-X)((1-X)^n - 1)}{(1-X)^n(X^n - X) + X^n(X-1)} \tag{2.26}$$

Die Abbildung 2.6 zeigt die Kennlinien der Squashfunktion mit unterschiedlichen Runlängen. Sie weist eine gute Anpassung der Sigmoidfunktion im Intervall $[-1, 1]$ auf.

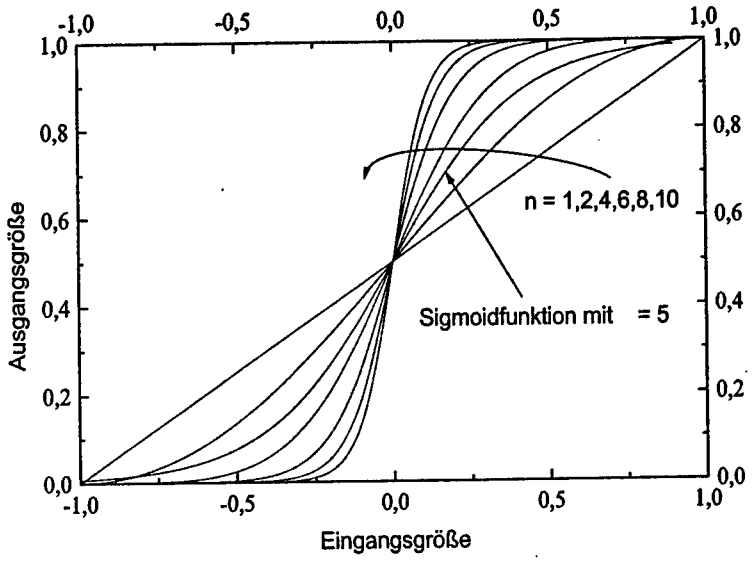


Abb. 2.6: Kennlinien der Squashfunktion mit unterschiedlichen Runlängen

Realisiert wurde diese Funktion mit zwei verbundenen sogenannten Runlängenakzeptoren, die sich mit Hilfe einer Klasse von stochastischen Automaten beschreiben lassen. Die Squashfunktion bildet die Überführungsfunktion dieses Automaten. Das einschlägige Zustandsübergangsdiagramm für den Automaten wird in Abbildung 2.7 dargestellt.

Es kann als Anleitung zum Entwurf des umsetzenden Schaltwerks dienen. Die Abbildung 2.8 stellt die Ergebnisse einer taktgenauen Software–Simulation für den Automaten dar. In der Darstellung wurden 1000 Punkte mit gleichem Abstand im Intervall $[-1, 1]$ abgetastet und zu je einer Bitfolge codiert, und die Ausgaben der Automaten mit unterschiedlichen Werten von n wurden dann in Blöcken von 1000 Bits decodiert. Für jeden Punkt auf der Achse des Eingangssignals wird der gleiche Vorgang dabei sechs Mal wiederholt, und alle sechs Ausgabewerte der S–Funktion werden dann gemittelt und als ein Punkt in der Kurve gezeichnet. Je größer der Wert n ist, um so steiler ist die Funktion, um so größer aber auch die stochastische Streuung, insbesondere in der Nähe des Ursprungs. Die negativen Auswir-

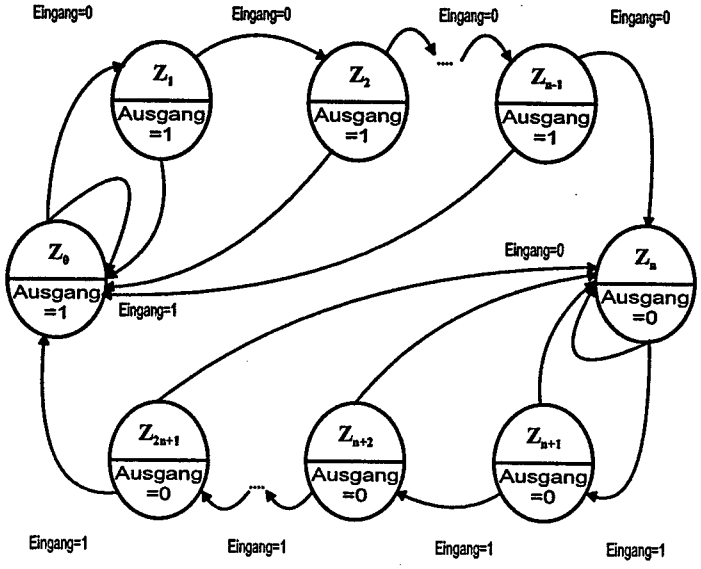


Abb. 2.7: Zustandsübergangsdiagramm der Squashfunktion

kungen solch einer großen Streuung in der Nähe des Ursprungs werden im nächsten Kapitel besprochen.

Für die Lernphase des BP–Algorithmus wird die erste Ableitung der nichtlinearen Funktion benötigt. Dafür wird hier ebenfalls ein stochastischer Automat verwendet, dessen Überführungsfunktion die notwendige Ableitung annähert. Diese Funktion ist ebenfalls eine gebrochene rationale Funktion und nie negativ im Intervall $[-1, 1]$. Sie wurde in [51] als Bogenfunktion bezeichnet (B–Funktion als Kurzform), weil sie bogenförmig ist. Die B–Funktion läßt sich im Wahrscheinlichkeitsraum folgendermaßen ausdrücken:

$$Y(X,n) = \frac{-(1-X)^{n+1} - X^{n+1} + 2(X^2 - X + 1)}{2(X^2 - X + 1)} \qquad (2.27)$$

Der theoretische Verlauf der Kennlinie der B–Funktion mit unterschiedlichen Werten von n wird in der Abbildung 2.9 gezeigt und die experimentellen Ergebnisse per taktgenauer Software–Simulation in der Abbildung 2.10. Da die B–Funktion nicht die ideale Ableitung der Squashfunktion ist, sollen die möglichen Auswirkungen dieses Unterschieds auf die Konvergenz des Verfahrens später untersucht und diskutiert werden.

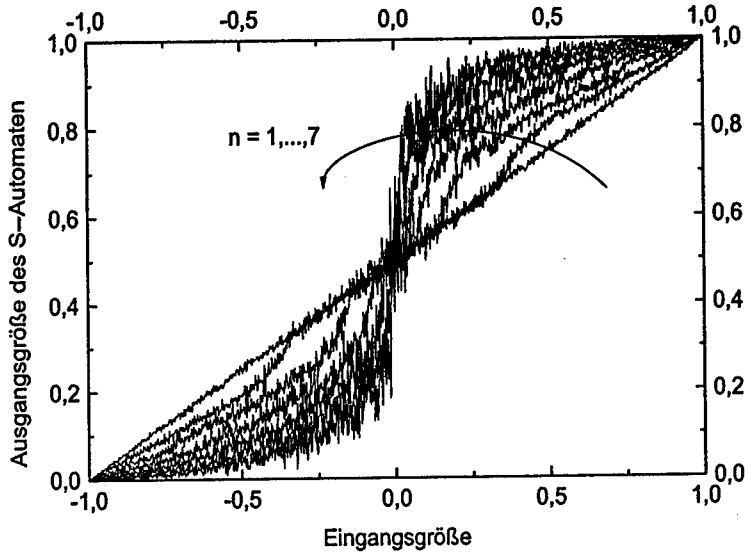


Abb. 2.8: Verlauf der Squashfunktion mit unterschiedlichen Runlängen bei taktgenauer Software–Simulation

Die zwei obigen Nichtlinearitäten bilden den Kern der Neuronen, welche die Informationen sowohl in der Lernphase als auch in der Arbeitsphase verarbeiten. Zum Speichern der zu verarbeitenden Informationen sind speichernde Glieder erforderlich, welche die Rolle der Gewichte in einem BP–Netz spielen sollen. Wie solche Glieder durch stochastische Rechentechnik implementiert werden und was bei der Implementierung für die Konvergenzeigenschaften des Netzes zu beachten ist, wird im kommenden Kapitel detailliert erläutert.

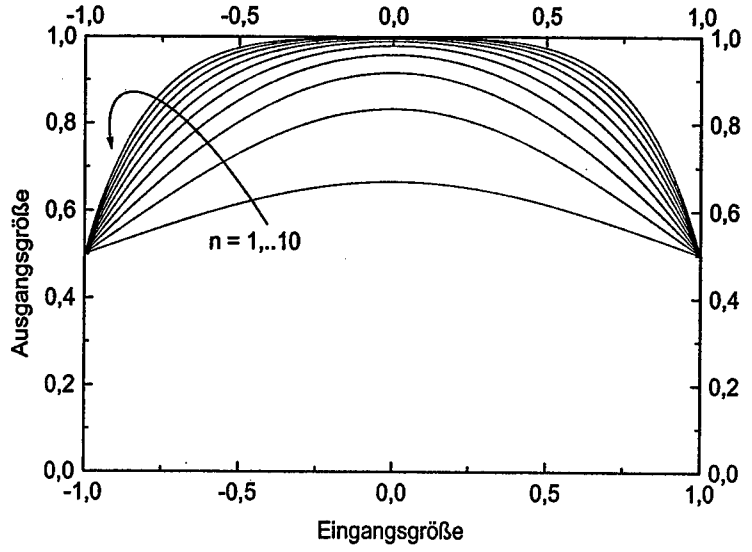


Abb. 2.9: Kennlinien der B–Funktion mit unterschiedlichen Runlängen

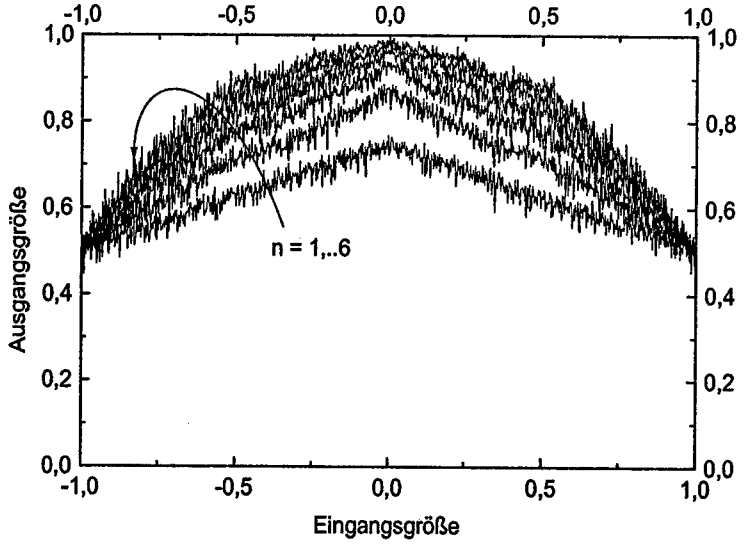


Abb. 2.10: Der per taktgenauer Software–Simulation ermittelte Verlauf der B–Funktion mit unterschiedlichen Runlängen

3 Parameterwahl und Lernverhalten

Die Parameterwahl ist für jeden Lernalgorithmus sehr wichtig und spielt häufig sogar die entscheidende Rolle für die Konvergenz des Lernvorgangs. Bedauerlicherweise ist sie aber meistens von den zu lösenden Problemstellungen abhängig. Über dieses Thema wurde in der Literatur oft in Verbindung mit konkreten Beispielen gesprochen. Eine allgemeine Regel, die für alle Fälle gilt, konnte bisher nicht gegeben werden, obwohl ein konventioneller BP–Algorithmus mit Impulsterm nur zwei Parameter besitzt, Lernrate γ und Momentum–Konstante α, und beide Parameter voneinander unabhängig sind. In dem vorgeschlagenen stochastischen Verfahren gibt es noch mehr Parameter als im konventionellen BP–Algorithmus. Im Gegensatz zu Lernrate und Momentum-Konstante im konventionellen Fall haben die Parameter gegenseitigen Einfluß aufeinander, weil sie die Genauigkeit der Gewichtespeicherung und die Größe der stochastischen Streuung beeinflussen können. Dies macht die Parameterwahl für das Training noch komplizierter. Aus diesem Grund ist eine Untersuchung zur Strategie und Methode der Parameterwahl notwendig.

In diesem Kapitel wird zuerst der Mechanismus zur Speicherung und Modifikation der Gewichte erläutert und danach eine 1:1 Abbildung zum konventionellen BP–Algorithmus hergestellt, um den Zusammenhang zwischen den Parametern des stochastischen Verfahrens und denen des konventionellen BP–Algorithmus zu verdeutlichen. Dadurch können die aus der Literatur bekannten Ergebnisse für die Wahl der Parameter im konventionellen BP–Algorithmus als Referenz oder Hinweise für den stochastischen Fall dienen. Anschließend werden die Strategie und zu beachtende Gesichtspunkte bei der Auswahl jedes einzelnen Trainingsparameters besprochen.

3.1 Gewichte

In einem BP–Netz sollen Gewichte in der Lernphase allmählich modifiziert werden, um eine optimale Wertebelegung zu garantieren. Danach müssen die gelernten Gewichte für die Arbeitsphase mit ausreichender Genauigkeit gespeichert werden. Dazu sind speichernde Glieder erforderlich, welche die obigen Aufgaben übernehmen.

3.1.1 INDIE–Glied

Wenn die beiden Seiten der Gleichung (2.7) über die Zeit im Intervall $[t_0, t_1]$ integriert werden, erhält man:

$$w_{ij}^{[l]}(t_1) - w_{ij}^{[l]}(t_0) = \int_{t_0}^{t_1} \gamma \delta_j^{[l]} x_i^{[l-1]} \, d\tau \quad j = 1, ..., n_l, \;\; i = 1, ..., n_{l-1} \quad (3.1)$$

Diese Gleichung entspricht einer gesamten Gewichtsänderung über das Intervall $[t_0, t_1]$. Das integrative Glied der stochastischen Rechentechnik (das sogenannte INDIE) [36] kann der Aufgabe gerecht werden. Abbildung 3.1 zeigt das Blockschema eines INDIE.

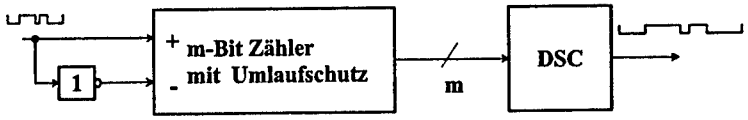


Abb. 3.1: Blockschema eines INDIE

Der m–stufige Zähler ist für das Speichern und die Modifikation der Gewichte zuständig. Er ändert sich bei jedem Takt, indem er beim Erscheinen einer Eins bzw. Null vorwärts bzw. rückwärts zählt. Um den Über– und Unterlauf des Zählers zu vermeiden, ist ein Mechanismus zur Umlaufsperre integriert. Dadurch kann die Gewichtsmodifikation über einen langen Zeitraum verfälscht werden. Diese Nebenwirkung auf die Konvergenz des Netztrainings wird in den folgenden Abschnitten diskutiert. Die Zählerlänge m bestimmt die Genauigkeit der Repräsentation der Gewichte und den Lernschritt des Trainings. Demnach gehört sie zu einem der zu untersuchenden Trainingsparameter. Sei X die Wahrscheinlichkeit für das Auftreten einer Eins in der Eingangsfolge des INDIE und innerhalb der Periode von N Takten stationär, dann läßt sich der aktuelle Zählerstand iz nach dem Ablauf der N Takte so beschreiben:

$$iz = iz_0 + (2X - 1) * N \quad (3.2)$$

Hier ist iz_0 der alte Zählerstand vor den N Takten und wegen der Umlaufsperren gilt folgende Ungleichung für iz_0:

$$0 \leq iz_0 + (2X - 1) * N \leq 2^m - 1. \quad (3.3)$$

Das aktuell im INDIE abgespeicherte Gewicht kann im W–Bereich wie folgt abgeschätzt werden:

$$\tilde{W} = \frac{iz}{2^m - 1} \quad (3.4)$$

3.1.2 ADDIE–Glied

Im konventionellen BP–Algorithmus mit Impulsterm ist die aktuelle Gewichtsänderung eine gleitende Mittelwertbildung zwischen der Multiplikation der Fehler mit der Aktivität des vorhergehenden Neurons und den bisherigen Gewichtsänderungen von vorherigen Lernschritten (siehe Gleichung (2.11)). Aus dieser Sicht ist die Speicherung der alten Gewichtsänderungen und die Erzeugung eines Bitstroms für die Gewichtsänderung erforderlich. Dafür ist das sogenannte adaptive Glied (ADDIE als Abkürzung in [36]) geeignet. Das ADDIE besteht ebenfalls aus einem Binärzähler mit Umlaufsperre und einem *DS*–Codierer. Zusätzlich beinhaltet es eine Rückkopplung, welche die Ausgangsbitfolge wieder auf seinen Eingang zurückführt. Abbildung 3.2 zeigt das Blockschema eines ADDIE mit einem n–stufigen Zähler.

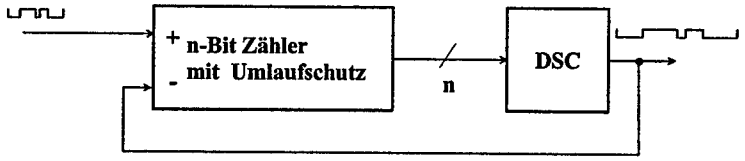


Abb. 3.2: *Blockschema eines ADDIE mit einem n–stufigen Zähler*

Der Zählerstand des n–stufigen Zählers wird ebenfalls in einen Bitstrom codiert, der sowohl als Ausgang für die weitere Verarbeitung bereitsteht als auch an den eigenen Eingang geführt wird. Dieser zurückgeführte Bitstrom dient dazu, den Zählerstand zu erniedrigen, wenn dort eine Eins anliegt. Liegen am Eingang und Ausgang gleiche Werte an, so wird der Zählerstand nicht verändert. Wird ein stationärer Wert in eine Eingangsbitfolge codiert, stellt sich nach langer Beobachtung ein Gleichgewichtszustand ein, d.h. die Wahrscheinlichkeit des Auftretens einer Eins ist für die Eingangs– und Ausgangsfolgen gleich. Dieser Wahrscheinlichkeit entspricht der durch den maximalen Wert $(2^n - 1)$ geteilte Zählerstand. Deswegen wird das ADDIE in [36] auch als gleitender Decodierer bezeichnet. Aus der Sicht der Nachrichtentechnik ist ein ADDIE ein digitales Tiefpaßfilter mit exponentieller Stoßantwort.
Im vorliegenden Fall ist der Gleichgewichtszustand des ADDIE während des Trainings von geringer Bedeutung. Stattdessen wird der Einschwingvorgang ausgenutzt. Sei X die Wahrscheinlichkeit der Eingangsfolge und az_0 der alte Zählerstand des ADDIE, dann ergibt sich der aktuelle Zählerstand az über eine Periode von N Takten mit der Taktperiode t_c (siehe [36]):

$$az(N * t_c) \quad = \quad X(2^n - 1)(1 - e^{-\lambda}) + az_0 e^{-\lambda}$$

$$\lambda = \frac{N}{2^{n-1}-1}\left(\frac{t_c}{t_{CA}}\right) \tag{3.5}$$

Nach Massen [36] wird t_{CA} die Taktfrequenz genannt, mit welcher der Zählerstand für die Decodierung eingelesen wird. Hier dient das ADDIE nicht als Decodierer, sondern als Speicherglied für die Gewichtsänderung; das Verhältnis zwischen t_c und t_{CA} spielt in diesem Fall folglich keine Rolle[1]. Deshalb wird in der folgenden Diskussion der Einfachheit halber $t_c = t_{CA}$ angenommen.
Wenn der Zählerstand durch den maximalen Wert des Zählers geteilt wird, ergibt sich die entsprechende Abschätzung der Wahrscheinlichkeit:

$$\tilde{Y} \stackrel{def}{=} \frac{az}{2^n-1} = X(1-e^{-\lambda}) + \tilde{Y}_0 e^{-\lambda} \tag{3.6}$$

Die Zählerlänge n (genauer gesagt die Anzahl der Binärstellen des Zählers) bestimmt sowohl die Genauigkeit der darin gespeicherten Gewichtsänderung als auch den Anteil der alten Gewichtsänderung für die aktuelle Gewichtsmodifikation bei einem Lernschritt. Demnach ist sie auch ein wesentlicher Parameter für den Trainingsvorgang.

3.1.3 Synapsenelement

Ein Synapsenelement besteht aus ADDIE und INDIE. Es bildet Verbindungen zwischen den Neuronenschichten. Das Synapsenelement dient dazu, daß Multiplikationen zwischen Gewicht, Neuronaktivität und zurückgeführtem Fehler und Gewichtsänderung auf der Bitstrom–Ebene berechnet werden, und daß ein Gewicht gespeichert und während der Lernphase modifiziert wird. Daher ist es für das *On–chip–learning* ein wesentliches Kernstück des Neurochips, das einen großen Teil der Chipfläche in Anspruch nimmt. Die Abbildung 3.3 zeigt den Aufbau eines Synapsenelements in seinen wichtigsten Einzelheiten.

3.2 Das Verhältnis zum konventionellen BP–Algorithmus

In diesem Abschnitt wird eine 1:1–Abbildung zwischen dem konventionellen BP–Verfahren und dem auf stochastischer Technik beruhenden Verfahren hergestellt, um daraus Erkenntnisse darüber zu gewinnen, wie und wieweit

[1] Bei der Decodierung kann eine Verfälschung der Wahrscheinlichkeit bei ungünstigem Verhältnis von t_c und t_{CA} auftreten [36]

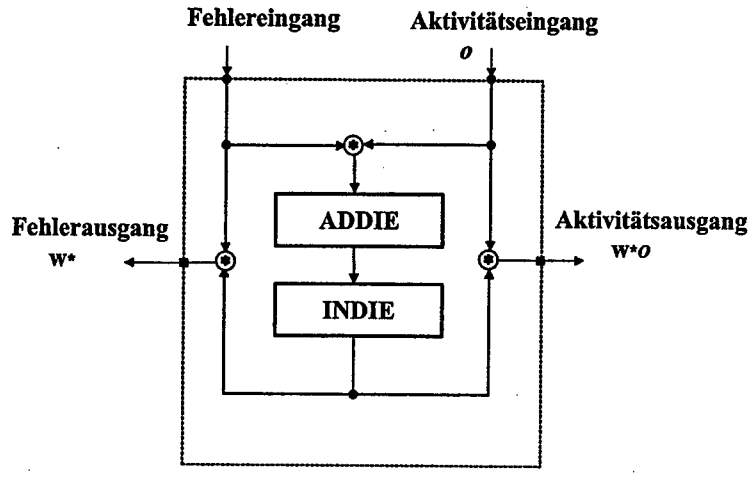


Abb. 3.3: Aufbau eines Synapsenelements

die neu auftauchenden Trainingsparameter (z.B ADDIE– und INDIE–Länge usw.) den Trainingsvorgang steuern können. Da Strategien zur Auswahl der Lernrate γ und der Momentum–Konstante α im konventionellen BP–Algorithmus der Literatur entnommen werden können und ihr Einfluß auf die Konvergenz des Trainings gut untersucht ist, können aus der Herstellung eines Zusammenhangs zwischen dem konventionellen BP–Algorithmus und dem hier vorgestellten Verfahren Rückschlüsse auf die Parameterwahl und deren Wirkung auf letzteres gezogen werden.

3.2.1 Eine 1:1–Abbildung

In der folgenden Herleitung wird nur das ***Online–Training*** betrachtet und die Präsentation eines Musterpaars soll N Takte dauern. Auch werden der Klarheit und Einfachheit halber alle Ober– und Unterindizes weggelassen. So ergibt sich die Gewichtsänderung je Lernschritt im konventionellen BP–Algorithmus mit Impulsterm:

$$\Delta w(k+1) = \gamma \delta x + \alpha \Delta w(k) \tag{3.7}$$

Wie in der Abbildung 3.3 gezeigt, ist der Eingang des ADDIE das Resultat der Multiplikation von δ und o, nämlich $x = \delta o$. Nach der Nomenklatur im vorigen Kapitel ergibt sich die Wahrscheinlichkeit der Eingangsfolge zu $X = \frac{1}{2}(x+1)$. Nach der Gleichung (3.6) läßt sich nun die Ausgabe Y des

ADDIE, welche wiederum der Eingabe des folgenden INDIE entspricht, im W–Bereich wie folgt beschreiben[2]:

$$Y = \frac{1}{2}(\delta o + 1)(1 - e^{-\lambda}) + Y_0 e^{-\lambda} \tag{3.8}$$

Nach der Gleichung (3.4) gewinnt man das neue Gewicht im W–Bereich für den aktuellen Lernschritt:

$$\begin{aligned} W(k+1) &= \frac{iz(k+1)}{2^m - 1} \\ &= \frac{iz(k) + (2Y-1)N}{2^m - 1} \\ &= W(k) + (2Y-1)\frac{N}{2^m - 1} \end{aligned} \tag{3.9}$$

$W(k)$ ist der Gewichtswert des k-ten Lernschritts. Aus der Gleichung (3.9) ist die Gewichtsänderung zwischen Lernschritt k und $k+1$ deutlich zu sehen, d.h.:

$$\begin{aligned} \Delta W(k+1) &\stackrel{def}{=} W(k+1) - W(k) \\ &= (2Y-1)\frac{N}{2^m - 1} \end{aligned} \tag{3.10}$$

Nach den Gleichungen (3.8) und (3.10) erhält man die entsprechende Beschreibung im M–Bereich:

$$\begin{aligned} \Delta w(k+1) &= w(k+1) - w(k) \\ &= [2W(k+1) - 1] - [2W(k) - 1] \\ &= 2(W(k+1) - W(k)) \\ &= 2\Delta W(k+1) \\ &= 2(2Y-1)\frac{N}{2^m - 1} \\ &= \frac{2N}{2^m - 1}[\delta o(1 - e^{-\lambda}) + (2Y_0 - 1)e^{-\lambda}] \\ &= \frac{2N}{2^m - 1}(1 - e^{-\lambda})\delta o + e^{-\lambda}\Delta w(k) \end{aligned} \tag{3.11}$$

Im Vergleich mit der Gleichung (3.7) erhält man die folgenden Gleichungen, welche zum Ausgangspunkt für die weitere Diskussion genommen werden können:

$$\alpha = e^{-\lambda} = e^{-\frac{N}{2^{n-1}-1}} \tag{3.12}$$

[2]ohne Berücksichtigung von Schätzfehlern

$$\gamma = \frac{2N}{2^m - 1}(1 - e^{-\lambda}) = \frac{2N}{2^m - 1}(1 - \alpha) \tag{3.13}$$

3.2.2 Vorschläge zur Parameterwahl

Die obigen Beziehungen zwischen den Lernparametern der beiden Verfahren geben eine Anregung für die Auswahl der Lernparameter des stochastischen Verfahrens. Aus den vorherigen Gleichungen sind folgende Schlüsse zu ziehen:

- Nicht nur durch die ADDIE–Länge n wird die Momentum-Konstante α gesteuert, wie man erwartet hat, sondern auch durch die Taktanzahl N. Jedoch hat die ADDIE–Länge n einen größeren Einfluß auf die Steuerung der Momentum-Konstante als die Taktanzahl N, weil der Term 2^n ein exponentielles Verhältnis zum Term n hat. Der Wert, den die Momentum-Konstante in diesem Fall annehmen kann, liegt im Intervall $[0, 1]$. Je größer n ist, desto größer ist das Momentum. Bei N ist es umgekehrt.

- Die Größe der Lernrate γ ist ebenfalls durch mehrere Parameter im stochastischen Verfahren steuerbar, nämlich durch die INDIE–Länge m, die ADDIE–Länge n sowie die Taktanzahl N. Dabei beeinflußt die INDIE–Länge m die Steuerung der Lernrate stärker als die übrigen Parameter.

- Die beiden Lernparameter γ und α sind nicht mehr unabhängig voneinander wie im konventionellen Fall. Das bedeutet, daß einige in der Literatur vorgeschlagenen Kombinationen von γ und α nicht in den stochastischen Fall übernommen werden können, weil sie die Gleichung (3.13) nicht erfüllen.

- Aus Sicht der Steuerung der Lernschrittweite ohne Berücksichtigung der stochastischen Streuung sind die INDIE–Länge m und die ADDIE–Länge n, welche sich nur in einem begrenzten Bereich der ganzen Zahlen befinden, die entscheidenden Parameter für die Konvergenz des Trainings. Dadurch könnte eine Strategie bei der Suche nach den optimalen Trainingsparametern so aussehen, daß der Versuch vorzugsweise mit einer niedrigen festen Taktanzahl (z.B. 1000) anfangen sollte, um damit den Suchvorgang zu beschleunigen.

3.3 Parameterwahl bei taktgenauer Bitstrom–Simulation

Bei der taktgenauen Bitstrom–Simulation handelt es sich um eine Software–Simulation, welche auf der Bitstrom–Ebene die in der Hardware ablaufenden Vorgänge des vorliegenden Verfahrens taktweise simuliert. Deshalb ist es sehr zeitaufwendig, die Simulation durchzuführen, was als Nachteil angesehen werden muß. Vorteilhaft ist jedoch die Exaktheit und Hardwarenähe des Verfahrens. Dadurch können die Konvergenzeigenschaften des Verfahrens realitätsnah genug und besser untersucht werden.
Die Wahl der Parameter ist von der Aufgabenstellung abhängig, d.h. für unterschiedliche Aufgaben kann es völlig unterschiedliche Kombinationen von Lernparametern geben, die das Training zur Konvergenz führen können. In diesem Abschnitt wird die Empfindlichkeit des Trainings auf die Wahl der Parameter, welche für das stochastische Verfahren relevant sind, durch eine taktgenaue Simulation untersucht. Dazu ist ein Maß erforderlich, mit dem die Trainingsergebnisse unter den verschiedenen Parameterkombinationen verglichen werden können. Die Trainingszeit ist dafür ein sinnvolles Maß, das der Anzahl der Epochen von Lernschritten entspricht, nach denen der Lernfehler relativ klein und innerhalb einer gewissen Genauigkeit akzeptabel ist. Wegen der stochastischen Eigenschaften des Verfahrens soll der gleiche Trainingsvorgang N_T mal wiederholt und eine durchschnittliche Anzahl $\tilde{N}_E$ von Epochen über N_T Trainingsvorgänge ermittelt werden. Für die Stopbedingung des Trainings ist ebenfalls ein durchschnittlicher Lernfehler $\tilde{E}$ zu ermitteln, der aus dem Lernfehler $\bar{E}_i$ (siehe Gleichung (2.4)) von N_F hintereinanderfolgenden Epochen geschätzt werden kann, d.h.:

$$\tilde{E} = \frac{1}{N_F} \sum_{i=1}^{N_F} \bar{E}_i \tag{3.14}$$

Wenn die folgende Bedingung für eine vorgegebene Konstante ε

$$\tilde{E} < \varepsilon \quad (\varepsilon > 0) \tag{3.15}$$

erfüllt wird, dann wird angenommen, daß das Netz mit einer gewissen Genauigkeit die Trainingsmenge bereits gelernt hat. Der Trainingsvorgang kann mit Erfolg abgeschlossen werden. Wenn die Ungleichung (3.15) nach N_N Epochen immer noch nicht erfüllt werden kann, wird der Trainingsvorgang zwangsläufig unterbrochen. Das Trainingsergebnis wird als Mißerfolg gewertet.
Um den Zeitaufwand der taktgenauen Simulation auf der Bitstrom–Ebene in tragbaren Grenzen zu halten, werden den obigen Konstanten folgende

Werte zugewiesen:

$$\varepsilon = 0{,}1$$
$$N_T = N_F = 8$$
$$N_N = 2000$$

3.3.1 ADDIE–Länge

Wie in 3.1.2 gesagt, entspricht das ADDIE einem gleitenden Decodierer. Sein Ausgang nähert sich im Lauf der Zeit seinem Eingang, wenn das Eingangssignal in der Zeit stationär ist. Es wird hier zunächst untersucht, wieviel Zeit oder genauer gesagt, wieviele Takte benötigt werden, um sich einem solchen Gleichgewichtszustand anzunähern, d.h. wie groß die Taktanzahl N sein muß, damit folgende Ungleichung für eine vorgegebene Zählerlänge n und eine kleine positive Konstante ε erfüllt wird:

$$\left|\tilde{Y} - X\right| < \varepsilon \tag{3.16}$$

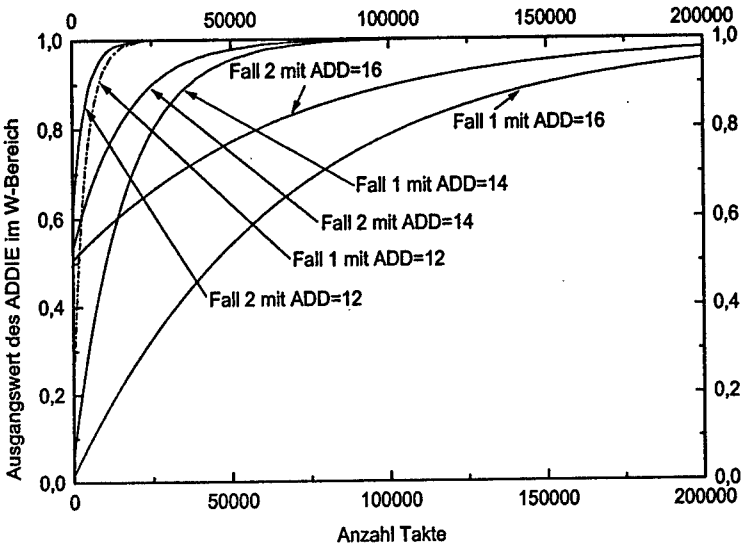
Durch Hinzufügen der Gleichung (3.6) in die obige Ungleichung ergibt sich:

$$\left|\tilde{Y} - X\right| = \left|\tilde{Y}_0 - X\right| e^{-\lambda} < \varepsilon \tag{3.17}$$

Wenn λ durch Gleichung (3.5) ersetzt wird, erhält man folgende Ungleichung:

$$N > (2^{n-1} - 1) \ln \frac{\left|\tilde{Y}_0 - X\right|}{\varepsilon} \tag{3.18}$$

Die Ungleichung (3.18) zeigt, wie viele Takte ein ADDIE als Decodierer braucht, bis seine Ausgabe die Eingangswahrscheinlichkeit unter vorgegebener Genauigkeit annähert. Im Extremfall, bei dem der alte Zählerstand des ADDIE der Wahrscheinlichkeit Null entspricht und die zu decodierende Wahrscheinlichkeit Eins zutrifft, sind mindestens 56581 Takte erforderlich, bis das Ergebnis eine Genauigkeit von 0,001 erreichen kann, wenn z.B. ein 14–stufiger Zähler im ADDIE verwendet wird. Die Zeit, welche das ADDIE zum Wechsel vom alten Zustand zum neuen Zustand benötigt, wird hier als *Anlaufzeit* bezeichnet. Die Abbildung 3.4 zeigt verschiedene Fälle für diese Anlaufzeit mit unterschiedlichen Zählerlängen des ADDIE.
Die Abbildung 3.4 deutet an: Bei kurzen ADDIEs stellt das ADDIE einen Stochastik–Digital–Decodierer dar und für lange ADDIEs wirkt es als Mittelwertbildner der zeitveränderlichen Wahrscheinlichkeit seiner Eingangsfolge. In diesem Fall schwingt sein Ausgang mit einem exponentiellen Übergangsverlauf auf jeden neuen Mittelwert ein. Diese Eigenschaft des ADDIE



Fall 1: Anfangswert=0.0, Zielwert=1.0 Fall 2: Anfangswert=0.5, Zielwert=1.0

Abb. 3.4: Simulationsergebnisse zur Anlaufzeit des ADDIE

wird hier für die Realisierung des Momentum im BP–Algorithmus ausgenutzt. Aus diesem Grund darf das ADDIE nicht zu kurz (kleiner als 12) gewählt werden[3]. Wenn das ADDIE zu kurz ist, dann spielt das Momentum keine Rolle mehr. In diesem Fall sollte das Training ohne ADDIE, d.h. ohne Momentum, durchgeführt werden, um mögliche Nebenwirkungen zu vermeiden (z.B. eine niedrige Genauigkeit für die Wertrepräsentation).

Die folgenden Abbildungen aus einigen behandelten Beispielen zeigen das Lernverhalten des Netzes zu den unterschiedlichen Werten von ADDIE–Längen. Obwohl die Parameterwahl von Problemstellungen abhängig ist, können die Ergebnisse aus diesen Beispielen als Leitfaden oder Strategie zur Parameterwahl für neue Aufgabenstellungen verwendet werden. In den Abbildungen repräsentiert die X–Achse die ADDIE–Länge und die Y–Achse die Durchschnittszyklen, die das Training bis zur Erfüllung der Stopbedingung benötigt. Wenn das Training nach maximaler Anzahl von Zyklen die Stopbedingung immer noch nicht erfüllen kann, wird das Ergebnis als „Divergenz" betrachtet. Natürlich könnte es vorkommen, daß die maximale Anzahl von Zyklen zu klein gewählt worden ist. So sollten hier die Anfüh-

[3]Aus Ergebnissen der Simulationen für viele Beispiele

rungsstriche verstanden werden, daß sich nach einer gewissen Anzahl von Lernschritten keine Konvergenz abzeichnet.

	Aufgabe 1	**Aufgabe 2**
Name	XOR	4–Bit–Encoder
Netztopologie	2-2-1	4-2-4
Musteranzahl	4	4
Taktanzahl	1000	1000
Maximale Zyklen	2000	2000
Stopbedingung	$\varepsilon = 0{,}15$	$\varepsilon = 0{,}15$
RUN / RUNAB	5/3	5/3

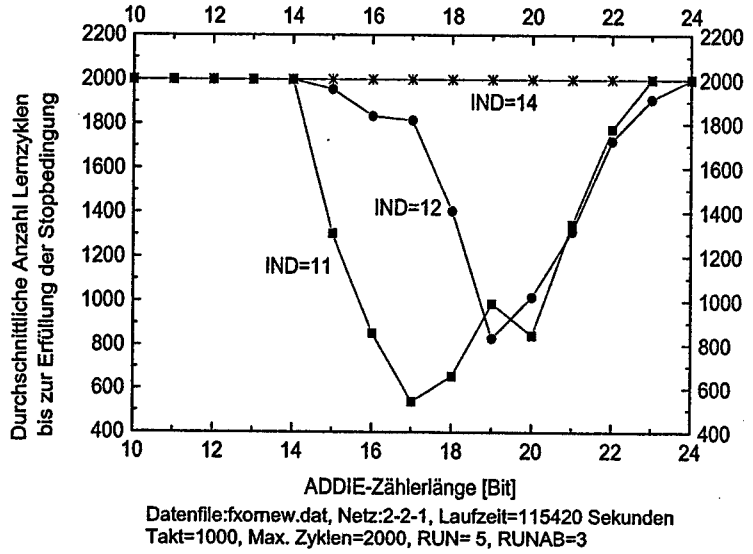


***Abb. 3.5:** Empfindlichkeit des Trainings bzgl. ADDIE bei Aufgabe 1*

Wenn eine Trainingsmethode sehr sensibel auf ihre Parameter reagiert, dann ist die Entscheidung für die richtige Parameterwahl nicht einfach zu treffen. Die Abbildung 3.5 zeigt, daß die hier diskutierte stochastische Methode nicht sehr empfindlich auf die Länge des ADDIE reagiert. Dies bedeutet, daß die Auswahl der ADDIE–Länge relativ elastisch ist. Ob die beste ADDIE–Länge gewählt wird, bestimmt nur die Geschwindigkeit der Konvergenz. Je einfacher die Aufgabe ist, desto kleiner ist der Einfluß des ADDIE auf die Konvergenz des Trainings, sofern das ADDIE nicht zu kurz (kleiner als 12) oder zu lang (länger als 24) gewählt wird. Die folgende Aufgabe 2 ist für ein MLP–Netz noch einfacher zu lösen als das XOR–Problem. Sie

wird in der Literatur häufig als Benchmark für das Training eines MLP–Netzes aufgenommen. Da diese Netztopologie mehr Ausgänge aufweist, ist der durchschnittliche Lernfehler größer als in der Aufgabe 1. So kann die Stopbedingung $\varepsilon = 0{,}1$ nie erfüllt werden, auch wenn das Training in der Tat schon konvergiert hat und das Netz ein gutes Ergebnis bei der Test–Phase liefern kann. Aus diesem Grund wird die Stopbedingung auf $\varepsilon = 0{,}15$ erhöht. Alle unter dieser Bedingung als „lernfähig" bezeichneten Netze liefern gute Ergebnisse bei der Test–Phase. Das Ergebnis von Aufgabe 2 bestätigt die Behauptung, daß die Auswahl des ADDIE bei dieser einfacheren Aufgabe noch beliebiger als zuvor ist.

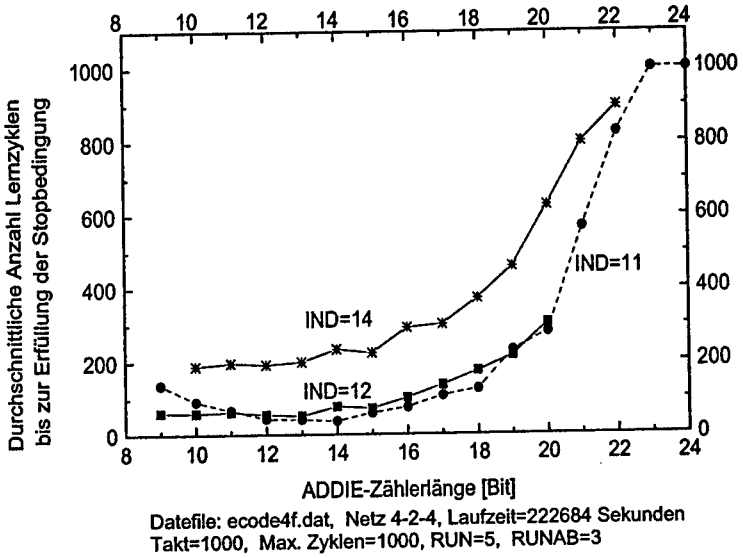


Abb. 3.6: Empfindlichkeit des Trainings bzgl. ADDIE bei Aufgabe 2

Die Kurven in den Abbildungen 3.5 und 3.6 wurden mit verschiedenen INDIE–Werten erhalten. Sie deuten auch an, daß die Auswahl des INDIE eine größere Rolle spielt als die des ADDIE. Im folgenden Abschnitt wird die Empfindlichkeit der Methode bezüglich der Auslegung des INDIE weiter untersucht.

3.3.2 INDIE–Länge

Die Anzahl der Zählerstellen des INDIE ist ein entscheidender Lernparameter, welcher die Konvergenz des auf stochatischer Technik basierenden

Lernverfahrens bestimmen kann. Dieser Parameter kann folgendermaßen den Lernvorgang stark beeinflussen:

- Die Größe der Lernrate γ (siehe Gleichung (3.13)) wird überwiegend von diesem Parameter kontrolliert. Wenn sich das INDIE um 1 Bit erhöht, wird die Lernrate γ quasi halbiert. Wegen der inhärenten und immer bestehenden stochastischen Streuung des Verfahrens darf die Lernrate γ nicht zu klein sein, sonst wird der Lernvorgang nicht von dem vorgegebenen Algorithmus, sondern von der stochastischen Streuung gesteuert. Um dies zu vermeiden, sollte die Lernrate γ folgende Bedingung erfüllen[4]:

$$\gamma > \sigma_{max} = \frac{1}{2\sqrt{N}} \tag{3.19}$$

  Wenn γ durch die Gleichung (3.13) ersetzt wird, ergibt sich eine Obergrenze für die Zählerlänge m des INDIE:

$$m < \frac{\ln\left(4N\sqrt{N}(1-\alpha)-1\right)}{\ln 2} \tag{3.20}$$

  Wenn keine Momentum-Konstante ($\alpha = 0$) auftaucht und 1000 Takte je Präsentation ($N = 1000$) vorgegeben werden, dann muß die Anzahl der Zählerstellen des INDIE nach der Ungleichung (3.20) kleiner als 17 sein. Sicher ist dies eine sehr konservative Obergrenze, weil die größte Streuung in der Ungleichung (3.19) verwendet wurde. Aus diesem Grund kann ein Gleichheitszeichen in die Ungleichung (3.20) eingesetzt werden, d.h.:

$$m \leq 17 \qquad wenn\ N = 1000\ und\ ohne\ Momentum - Konstante \tag{3.21}$$

  Ähnlich gilt auch:

$$m \leq 20 \qquad wenn\ N = 4000\ und\ ohne\ Momentum - Konstante \tag{3.22}$$

- Die Zählerlänge des INDIE bestimmt auch die Genauigkeit eines Gewichts, welches im INDIE gespeichert wird. Aus dieser Sicht sollte das INDIE lang genug gewählt werden, um eine gewisse Genauigkeit zu erreichen. Der BP–Algorithmus verlangt eine hohe Genauigkeit sowohl beim Speichern als auch beim Berechnen der Gewichte. Es ist

[4]Hier wird die mögliche Vergrößerung der Streuung durch die Nichtlinearität nicht berücksichtigt.

kaum vorstellbar, daß die Genauigkeit für die Speicherung der Gewichte niedriger als 0,01 wäre. So muß m folgende Ungleichung erfüllen:

$$m > 6 \tag{3.23}$$

- Wegen der Umlaufsperren des INDIE kann die Gewichtsmodifikation, welche während des Trainings stattfindet, verfälscht werden, wenn das INDIE zu kurz gewählt wird. Nun wird untersucht, wann solche Verfälschungen vorkommen können, falls die Dauer einer Präsentation N festgelegt wird. Die Gewichtsmodifikation iz läßt sich folgendermaßen ausdrücken:

$$iz = \begin{cases} 0 & wenn \quad iz_0 + (2X-1) * N < 0 \\ 2^m - 1 & wenn \quad iz_0 + (2X-1) * N > 2^m - 1 \\ iz_0 + (2X-1) * N & sonst \end{cases} \tag{3.24}$$

Der Überlauf des INDIE kann nur vorkommen, wenn die Wahrscheinlichkeit seiner Eingangsfolge größer als 0,5 ist, d.h., wenn $X > 0{,}5$, könnte folgende Situation entstehen:

$$iz_0 + (2X-1) * N > 2^m - 1 \tag{3.25}$$

Sei $X_0 = \frac{iz_0}{2^m-1}$ das alte Gewicht, welches nach der letzten Repräsentation im INDIE gespeichert wird. Dividieren beider Seiten der Ungleichung (3.25) durch $2^m - 1$ ergibt:

$$2^m < \frac{N(2X-1)}{1-X_0} + 1 \tag{3.26}$$

Die Ungleichung sagt aus, daß dann ein Überlauf eintritt, wenn die Länge des INDIE diese Ungleichung erfüllt. Um den Überlauf möglichst zu vermeiden, sollte das INDIE lang genug sein, nämlich:

$$m \geq \log_2\left(\frac{N(2X-1)}{1-X_0} + 1\right) \qquad wenn \quad X > 0{,}5 \tag{3.27}$$

Daraus erhält man für das INDIE eine Untergrenze, welche von der Dauer N einer Präsentation, der Wahrscheinlichkeit X seiner Eingangsfolge und dem alten gespeicherten Gewicht X_0 abhängig ist. Je dichter das alte Gewicht X_0 am Spitzenwert 1 liegt, umso wahrscheinlicher ist ein Überlauf, wenn m und N fest sind. So ist es sinnvoll

und vernünftig, einen Durchschnittswert, nämlich $X_0 = 0{,}5$, für die Aussage über eine konkrete Obergrenze des INDIE zu nehmen. Während des Trainings ist die Gewichtsänderung per Präsentation relativ klein, d.h sie schwingt in der Nähe des Ursprungs des Wertebereiches der Maschinenvariablen. Dies entspricht einer Schwingung um 0,5 im W–Bereich. Aus diesem Grund ist es vertretbar, eine konkrete Untergrenze für m mit dem Wert $X = 0{,}55$ zu ermitteln. Man erhält:

$$m \geq 8 \quad \textit{wenn} \quad N = 1000, \quad X_0 = 0{,}5 \quad \textit{und} \quad X = 0{,}55 \tag{3.28}$$

oder

$$m \geq 10 \quad \textit{wenn} \quad N = 4000, \quad X_0 = 0{,}5 \quad \textit{und} \quad X = 0{,}55 \tag{3.29}$$

Für den Unterlauf kann man durch eine ähnliche Analyse die gleiche Untergrenze für die Zählerlänge des INDIE erhalten. Aus der Ungleichung (3.27) ist klar ersichtlich, daß ein Überlauf bestimmt stattfinden wird, wenn das alte Gewicht bereits an der oberen Grenze des M–Bereiches liegt ($X_0 \approx 1{,}0$) und über die zurückgeführte Fehlerinformation weiter erhöht werden muß ($X > 0{,}5$). In dem Fall wird sich wegen der Umlaufsperre im INDIE–Zähler eine Verfälschung der Gewichtsmodifikation ergeben. Dies ist mit der Wertebeschränkung auf den M–Bereich $[-1, 1]$ verbunden. Es ist auch unmöglich, die zu lernenden Gewichte vor dem Training durch einen Faktor zu normieren, so daß sie während des Trainings garantiert im M–Bereich liegen werden. Über dieses Thema wird in den folgenden Kapiteln weiter gesprochen.

Die Ergebnisse einer Simulation für die Untersuchung der Empfindlichkeit des Verfahrens bezüglich der INDIE–Länge werden in den Abbildungen 3.7 und 3.8 gezeigt.

Aus den Abbildungen können die vorherigen Behauptungen über die Ober– und Untergrenze des INDIE bestätigt werden. Dies könnte für unbekannte Aufgabenstellungen ein Hinweis sein, wie man nach den richtigen Lernparametern suchen sollte. Die Abbildungen zeigen auch, daß die Lernverfahren mit dem stochastischen Rechenwerk noch empfindlicher auf den Parameter INDIE–Länge (steile Kurve) als auf die ADDIE–Länge (relativ flache Kurve) reagieren. Je komplexer die zu lernende Aufgabe ist, desto genauer muß das INDIE gewählt werden. Bei unbekannten Aufgaben könnte der hier gefundene beste Wert des INDIE für die Suche nach dem geeigneten INDIE als Ausgangspunkt und die hier erläuterten Ober– und Untergrenzen als Hinweis genommen werden.

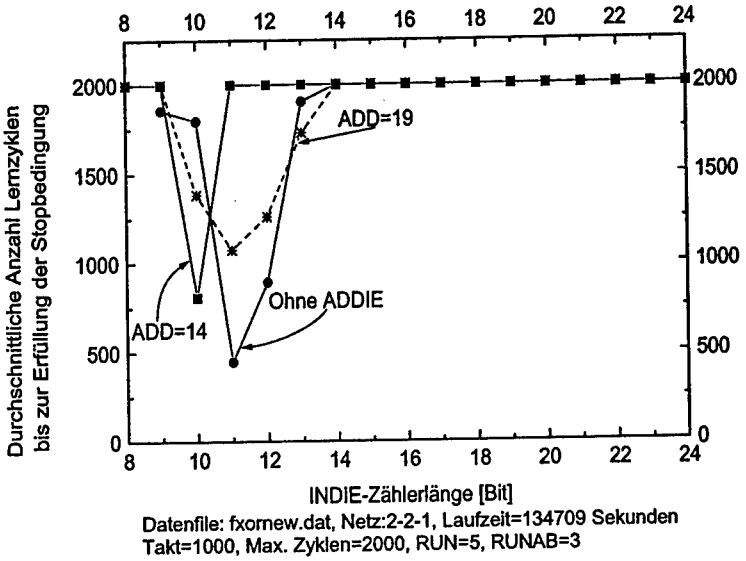


Abb. 3.7: Empfindlichkeit des Trainings bzgl. INDIE bei Aufgabe 1

3.3.3 Taktanzahl je Präsentation

Die Taktanzahl N je Präsentation beschreibt deren Dauer und ist ebenfalls ein wesentlicher Parameter, welcher den Trainingsvorgang folgendermaßen beeinflussen kann:

- Die Dauer des Trainings wird durch diesen Parameter bei der Software–Simulation stark beeinflußt. Je höher die Taktanzahl N ist, desto länger dauert das Training, bis das Netz konvergiert (falls überhaupt möglich).

- Die Größe der stochastischen Streuung ist durch diesen Parameter steuerbar (siehe Gleichung (2.17)). Je größer N ist, umso kleiner ist die entsprechende Streuung. Dadurch kann die Rechengenauigkeit während der Datenverarbeitung erhöht werden. Aber wegen der nichtlinearen Operation der Squashfunktion hat dies einen sehr geringen Einfluß.

- Die Größe von Lernrate γ und Momentum-Konstante α kann über den Wert von N eingestellt werden (siehe Gleichungen (3.13) und (3.12)). Die Lernrate γ ist proportional zur Taktanzahl N, wenn die Längen von INDIE und ADDIE fest sind. Bei der Momentum-Konstante α

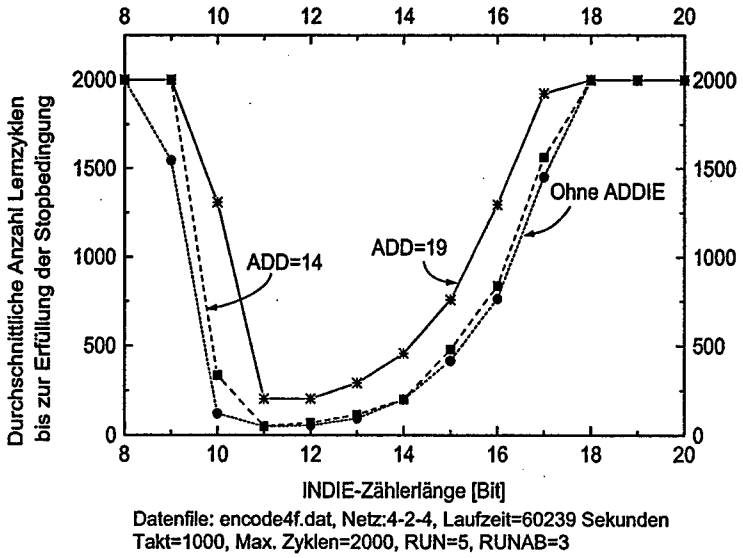


Abb. 3.8: Empfindlichkeit des Trainings bzgl. INDIE bei Aufgabe 2

ist es umgekehrt. Aus den Gleichungen (3.12) und (3.13) ist zu sehen, daß die Lernrate γ und die Momentum-Konstante α quasi unverändert bleiben, wenn sich die Taktanzahl N verdoppeln oder halbieren läßt und INDIE und ADDIE gleichzeitig um ein Bit erhöht oder verkürzt werden.

- Die Größe von N kann bei festem Wert der Zählerlänge des INDIE eine Ursache der Verfälschung der Gewichtsmodifikation sein, denn das INDIE ist eigentlich ein Integrator und N bestimmt die Dauer der Integration. Wenn die Integration zu lange dauert, kommt ein Umlauf im INDIE–Zähler bestimmt vor. In diesem Fall könnte die Ungleichung (3.26) als Hinweis für die Auslegung der Zählerlänge genommen werden, um die Verfälschung von Gewichtsmodifikationen zu vermeiden.

Für die Aufgabe 1 wurde eine Simulation mit unterschiedlichen Werten von N durchgeführt. Wegen des großen Zeitaufwands und zum besseren Vergleich wurde nur ein Netz ohne ADDIE in Betrieb gesetzt. Die Abbildung 3.9 zeigt das Ergebnis der Simulation. In dieser Abbildung ist deutlich zu sehen, daß sich die besten Werte für die Zählerlänge des INDIE, die zur kürzesten Trainingszeit führen, um ein oder zwei Bit erhöhen, wenn N ver-

doppelt oder vervierfacht wird. Dies bestätigt die obige Behauptung. Aus diesem Grund sollte man bei der Suche nach der besten Kombination von Parametern immer mit einem niedrigen Wert von N anfangen, um möglichst viel Zeit zu sparen. Andererseits sollte N zur Feineinstellung des Lernvorgangs verwendet werden, sofern die Hardware dies zuläßt. Als eine Faustregel sollte der Wert von N zwischen 1000 und 2000 liegen. Die Änderung von N sollte bei der Suche nach dem besten Wert eine gewisse Mindestgrößenordnung haben, weil eine kleine Änderung (z.B kleiner als 100) von N kaum eine Wirkung auf den Trainingsvorgang hat.

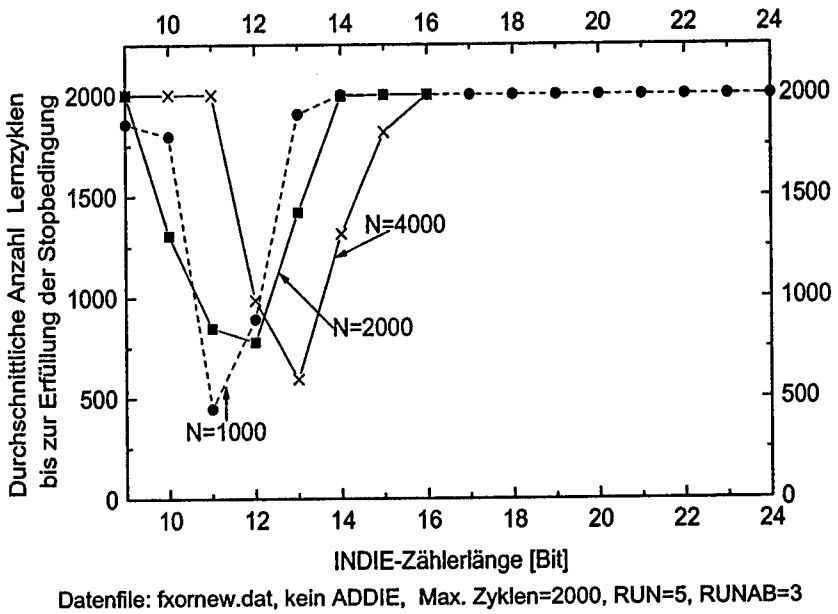


Abb. 3.9: Empfindlichkeit des Trainings bzgl. INDIE bei Aufgabe 1 mit unterschiedlicher Taktanzahl

3.3.4 Runlänge der stochastischen Automaten

In den obigen Abbildungen tauchen noch zwei weitere Parameter auf, nämlich RUN und RUNAB. Sie stehen für die Runlängen der stochastischen Automaten zur Bildung der Squash– bzw. B–Funktion. Die beiden Parameter bestimmen die Steilheit und Form dieser beiden Funktionen. Ihre gemeinsame Auswirkung auf den Lernvorgang entspricht dem μ der Sigmoidfunktion

im konventionellen BP–Algorithmus (siehe Abschnitt 2.2.2.2). Im konventionellen Fall wird meistens $\mu = 1$ verwendet, denn wegen der Eigenschaft der Sigmoidfunktion $s(x)$, nämlich $s'(x) = \mu s(x)[1 - s(x)]$, wirkt μ wie ein Multiplikator der Lernrate γ (siehe Gleichungen (2.7) und (2.8)). Darum wurde der Wert von μ in der Literatur kaum in Betracht gezogen. Im Gegensatz dazu spielen die Runlängen der stochastischen Automaten hier eine größere Rolle im Trainingsvorgang als ihr Äquivalent im konventionellen Verfahren. Der Grund liegt bei den Einschränkungen, welche beim stochastischen Verfahren inhärent bestehen und im nächsten Kapitel im einzelnen diskutiert werden. Dort wird die Auswahl dieser beiden Parameter weiter besprochen und untersucht. Die hier verwendeten Werte von RUN und RUNAB (z.B. $RUN = 5$ und $RUNAB = 3$) sind durch viele Software–Simulationen gewonnen worden. Eine Squashfunktion mit $RUN = 5$ ist so steil wie eine Sigmoidfunktion mit $\mu = 8$. Dies wird in der Abbildung 3.10 gezeigt.

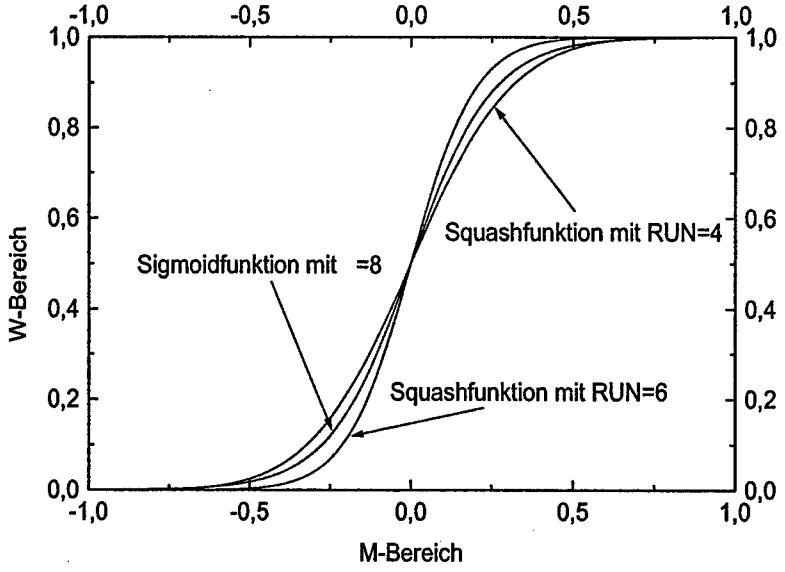


Abb. 3.10: *Vergleich der Steilheit zwischen Sigmoid– und Squashfunktion*

In der obigen Untersuchung wurde ein Geltungsbereich der Lernparameter durch qualitative Analyse erworben, der als Leitfaden der Auswahl von Trainingsparametern für unbekannte Aufgabenstellungen benutzt werden kann. Außerdem wiesen die Ergebnisse der Simulation gleichzeitig die Tat-

sache auf, daß der Trainingsablauf unter Verwendung des Verfahrens sehr empfindlich auf die Zählerlänge des INDIE ist. Mit einem falsch gewählten Wert kann das Training zur Divergenz führen. Diese Eigenschaft kann die Einsetzbarkeit des Verfahrens praktisch zunichte machen, weil sich die Suche nach einer richtigen Kombination der Lernparameter für das Training des Netzes bei unbekannten Lernaufgaben ebenso kompliziert wie die ursprüngliche Aufgabenstellung gestalten könnte. Außerdem ist eine beliebige Auswahl der Lernparameter bei der Hardware–Implementierung des Verfahrens nicht realistisch, insbesonders während der Trainingsphase. Was ist die Ursache dafür? Wie kann das Verfahren in dieser Richtung verbessert werden? Diese Fragen sollen im folgenden Kapitel untersucht und beantwortet werden.

4 Einschränkungen und Gegenmaßnahmen

In diesem Kapitel werden die Unterschiede zwischen dem hier behandelten Verfahren und dem konventionellen BP–Algorithmus bzw. zunächst die Schwachpunkte erläutert. Ihre Auswirkungen auf die Konvergenzeigenschaften des NN werden dann näher untersucht. Zum Schluß werden Vorschläge für Maßnahmen gegen negative Auswirkungen gemacht und diskutiert.

4.1 Unterschiede zum konventionellen BP–Algorithmus

Das hier vorgestellte Verfahren ist nichts anderes als eine Variante des klassischen BP–Algorithmus, welche auf der Basis der stochastischen Rechentechnik implementiert werden kann. Im Gegensatz zum konventionellen BP–Algorithmus tauchen hier Einschränkungen auf, welche die Konvergenzeigenschaften des stochastischen Verfahrens stark beeinflussen können:

- **Wertebereichseinschränkung**
  Der Informationsträger in diesem Verfahren ist die Wahrscheinlichkeit. Dies führt zu einer Wertebereichseinschränkung auf [0,1] für alle an der Verarbeitung beteiligten Datengrößen. Sie wird in den folgenden Diskussionen als $[0, 1]$–Einschränkung bezeichnet. Um dieser zu genügen, muß jede Problemvariable R vor der Verarbeitung durch einen Normierungsfaktor P_{max} in den M–Bereich transformiert werden (siehe [51]). Der Normierungsfaktor P_{max} ist problemabhängig und sollte gleich dem größten, vorzeichenlosen Betrag aller Datengrößen im entsprechenden Netz sein. Vor dem Training ist die Größenordnung der zu lernenden Gewichte unbekannt. So ist der Wert von P_{max} lediglich eine Abschätzung. Die Umlaufsperren im INDIE dienen dazu, daß die gelernten Gewichte zwingend im M–Bereich bleiben müssen, obwohl die optimalen Gewichte wegen einer möglichen falschen Abschätzung von P_{max} außerhalb des M–Bereiches liegen können. In einem solchen Fall kann die beste Gewichtsbelegung im Training nicht gefunden werden.

- **Das $\frac{1}{N}$–Mittelungsverfahren**
  Statt der arithmetischen Additionen werden mittelnde Additionen für

die Summation der gewichteten Neuroneneingänge verwendet. Diese Einschränkung entspricht der Verwendung der stochastischen Rechentechnik, weil das Resultat einer Addition den W–Bereich nicht überschreiten darf.

- **Squash– und Bogenfunktion**
  Statt der herkömmlichen Sigmoidfunktion und ihrer Ableitungsfunktion wird im Neuron die S– und die B–Funktion verwendet. Die B–Funktion entspricht nur einer groben Näherung der Ableitung der S–Funktion.

- **Stochastische Streuungen**
  Im Gegensatz zum konventionellen Fall, bei dem stochastische Streuungen von Menschenhand mit Absicht eingeführt werden und deren Stärke demnach kontrollierbar ist, sind diese hier verfahrensbedingt und können sich durch hintereinandergeschaltete stochastische Rechenwerke und Automaten für die Nichtlinearitäten massiv ausbreiten und unerfreulich vergrößern. Es wurde bereits im konventionellen Fall gezeigt, daß ein wenig stochastische Streuung dem Lernvorgang helfen kann, wenn das Training in einem lokalen Minimum steckenbleibt. Aber wird die Streuung zu stark und übernimmt sie eine dominierende Rolle beim Training, dann führt ein Training durch Backpropagation zu keinem sinnvollen Ergebnis.

- **Gegenstrom–Verfahren**
  Zugunsten der Hardware–Implementierung finden Lern- und Arbeitsphase während des Trainings gleichzeitig statt, d.h. bei jedem Takt wird ein Bit der Netzaktivität[1] erzeugt und gleichzeitig eine Gewichtsmodifikation um ein Bit durchgeführt. Dies hat zur Folge, daß die zu codierende Maschinenvariable nicht mehr stationär bleibt.

4.2 Auswirkungen auf die Konvergenz

Hier wird zunächst untersucht, welche Auswirkungen die oben genannten Abweichungen vom originalen BP–Algorithmus auf die Konvergenz des Trainings haben. Es ist vernünftig und logisch, zuerst zu studieren, wie sich ein BP–Netz in konventioneller Rechentechnik verhält, wenn die oben genannten Einschränkungen einzeln oder alle gleichzeitig wirken. Die so gewonnenen Ergebnisse können als asymptotische Eigenschaften, welche in einem

[1] in Form einer Bitfolge

BP–Netz mit stochastischen Rechenwerken erst nach unendlich langer Beobachtungszeit sichtbar werden, für den stochastischen Fall genutzt werden. Bei einer zeitlich begrenzten Beobachtung kann sich ein Netz in stochastischer Rechentechnik anders verhalten. Das ist auch ein Grund, warum eine taktgenaue Software–Simulation für diese Untersuchung erforderlich ist.

4.2.1 BP–Netz in konventioneller Rechentechnik

Für die Untersuchungen wurde ein Simulationsprogramm erstellt, welches alle Fälle der obengenannten Einschränkungen behandeln kann. Zwei Sorten von künstlichem Rauschen wurden im Programm berücksichtigt, nämlich Rauschanteile für die Gewichte und das Eingabesignal des Neurons. Die Stärke des Rauschens wird durch zwei Parameter W_r und X_r gesteuert, welche durch eine Benutzerschnittstelle (*Dialog Box*) vor dem Training gesetzt oder während des Trainings geändert werden können. Die Rauschanteile werden den Trainingsvorgang folgendermaßen beeinflussen:

- Bei der Gewichtskorrektur:
  $W(k+1) = W(k) + \Delta W(k+1) + r_1 * W_r \qquad r_1 \in [-1,1]$
- Beim Berechnen der Ableitung der Sigmoidfunktion:
  $s'(x) = \mu s(x)[1 - s(x)] + r_2 * x_r \qquad r_2 \in [-1,1]$

r_1 und r_2 werden durch den Zufallsgenerator des Compilers erzeugt. Daher ist die genaue Betrachtung des Einflusses der stochastischen Streuungen und insbesondere des Gegenstrom–Verfahrens in einem BP–Netz mit konventioneller Rechentechnik kaum zu realisieren. Dies wird deshalb in einem Netz mit stochastischer Rechentechnik näher betrachtet (Siehe Abschnitt 4.2.2).

4.2.1.1 [0,1]–Einschränkung

In der konventionellen Rechentechnik können alle Datengrößen vorzeichenbehaftet sein, doch soll hier die [0,1]–Einschränkung eine Wertebereichseinschränkung der beteiligten Datengrößen auf $[-1,1]$ bedeuten, d.h. sich auf den M–Bereich beziehen. Hier handelt es sich nur um die Gewichte des Netzes, weil die Trainingsmuster vor dem Training auf den M–Bereich normiert werden können. Wenn die Gewichte am Anfang innerhalb des M–Bereiches initialisiert werden, dann ist eine Überprüfung für den Umlauf des Wertebereiches nur bei den Additionsoperationen notwendig, nämlich bei der Gewichtskorrektur und bei den Skalarprodukten der Vektoren. Dies kann erreicht werden, indem ein Normierungsfaktor P_{max} vorher abgeschätzt wird und die zu lernenden Gewichte entsprechend normiert werden, d.h.:

$$\hat{w}_{ij} \stackrel{def}{=} \frac{w_{ij}}{P_{max}} \tag{4.1}$$

Der Normierungsfaktor P_{max} sollte groß genug sein, so daß kein Überschreiten des Wertebereiches von Gewichten bei der Gewichtskorrektur stattfinden kann. Bezeichnet $\hat{\mu}$ die Steilheit der Sigmoidfunktion in diesem Fall, so ergibt sich die Aktivität des Neurons (siehe Gleichung 2.5 im Abschnitt 2.2.2.2) zu:

$$\begin{aligned} x_j^{[l]} &= s(\sum_{i=0}^{n_{l-1}} \hat{w}_{ji}^{[l]} x_i^{[l-1]}) \\ &= \frac{1}{1+e^{-\hat{\mu}\sum_{i=0}^{n_{l-1}} \hat{w}_{ji}^{[l]} x_i^{[l-1]}}} \\ &= \frac{1}{1+e^{-\frac{\hat{\mu}}{P_{max}}\sum_{i=0}^{n_{l-1}} w_{ij}^{[l]} x_i^{[l-1]}}} \end{aligned} \tag{4.2}$$

Im Vergleich zur ursprünglichen Definition der Aktivität des Neurons ergibt sich:

$$\hat{\mu} = P_{max}\mu \tag{4.3}$$

Aus Gleichung 4.3 läßt sich folgendes schließen: Erstens kann die Auswirkung des Normierungsfaktors P_{max} durch den Lernparameter μ, d.h. die Steilheit der Sigmoidfunktion, kompensiert werden, obwohl er nicht vorher angegeben werden kann. Zweitens sollte eine steilere Sigmoidfunktion (mit großem Wert von μ, weil P_{max} meistens größer als Eins ist) verwendet werden, um ein erfolgreiches Training zu erreichen, wenn die [0,1]–Einschränkung auf den Lernvorgang Einfluß nimmt. Wie in Abschnitt 3.3.4 erläutert, wirkt μ wie ein Multiplikator der Lernrate γ (siehe Gleichungen (2.7) und (2.8)). Daher muß die Lernrate γ in diesem Fall kleiner als zuvor gewählt werden. Sonst wird das Trainingsverhalten divergent. Abbildung 4.1 zeigt das Simulationsergebnis eines konventionellen BP–Algorithmus und Abbildung 4.2 das eines BP-Algorithmus mit der [0,1]–Einschränkung. Interessant ist, daß die Steilheit der Sigmoidfunktion ($\mu = 8$), mit welcher das Training konvergiert, der Steilheit der S–Funktion mit $RUN = 5$ entspricht. Diese Übereinstimmung wurde bereits im Abschnitt 3.3.4 gefunden. Insofern könnte das hier erhaltene Ergebnis ein indirekter Hinweis für die Auswahl RUN=5 sein, die durch viele Simulationen in der stochastischen Rechentechnik für die Aufgabe 1, nämlich das XOR–Problem, gewonnen wurde. Aufgefallen ist auch, daß die Rauschanteile im Fall der [0,1]–Einschränkung die Konvergenz des Trainings stark beeinflussen können. Diese Eigenschaft kann sich einerseits als Vorteil eines Netzes in stochastischer Rechentechnik auswirken, andererseits ist der Trainingsvorgang schwer zu steuern. Mit gewissen Rauschanteilen können die Anfangswerte

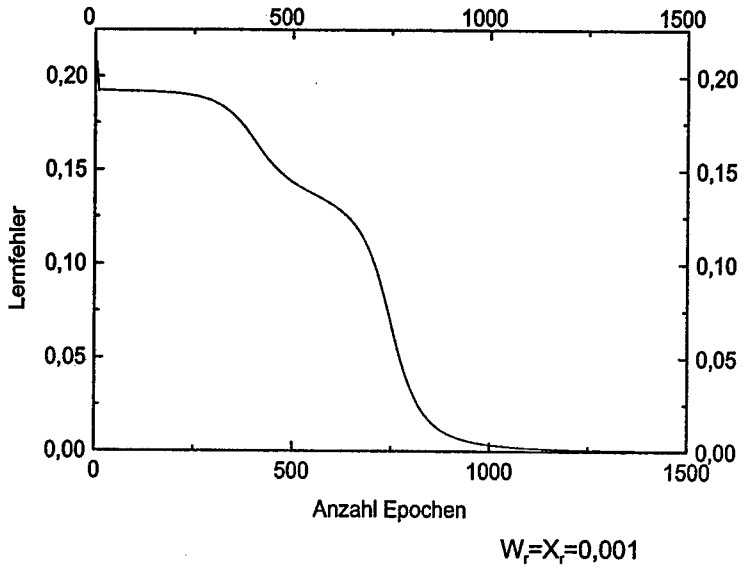


Abb. 4.1: Ergebnis des konventionellen BP für das XOR–Problem

aller Gewichte auf Null gesetzt werden, was bei einigen Aufgaben den Konvergenzvorgang beschleunigen könnte. Bei der Hardware–Implementierung des hier vorgestellten Verfahrens wird eine Null–Initialisierung aller Gewichte bereits verwendet.

Die obige Analyse deutet an, daß die [0,1]–Einschränkung allein kein großes Hindernis für die Konvergenz des Verfahrens darstellen dürfte, falls eine steilere Sigmoidfunktion und bestimmte Rauschanteile zum Einsatz gebracht werden. Die Rauschanteile bringen zudem noch den Vorteil, daß alle Gewichte mit Null initialisiert werden können, ohne daß dadurch das Netz seine Fähigkeit verliert, die Gewichte durch BP zu adaptieren. Diese einfache Initialisierung der Gewichte kommt einer Hardware–Implementierung entgegen, obwohl es keine Garantie gibt, daß eine Null–Initialisierung den Trainingsvorgang bei allen Aufgabenstellungen beschleunigen kann.

4.2.1.2 $[\frac{1}{N}]$–Problem

Das $[\frac{1}{N}]$–Problem stammt aus der [0,1]–Einschränkung, die durch eine traditionelle Addition von N Summanden jedoch verletzt werden kann. Um dies zu vermeiden, werden statt arithmetischer Additionen mittelnde Additionen verwendet. Sei $\tilde{\mu}$ die Steilheit für diesen Fall, dann läßt sich die

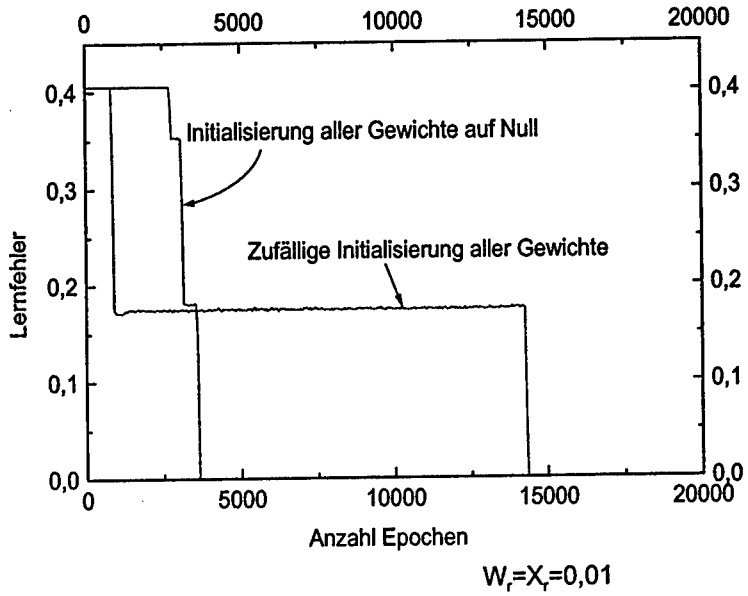


Abb. 4.2: Ergebnis des BP mit der [0,1]–Einschränkung für das XOR–Problem

Aktivität eines Neurons folgendermaßen beschreiben:

$$\begin{aligned} x_j^{[l]} &= s\left(\frac{1}{N}\sum_{i=0}^{n_{l-1}} w_{ij}^{[l]} x_i^{[l-1]}\right) \\ &= \frac{1}{1+e^{-\frac{\mu}{N}\sum_{i=0}^{n_{l-1}} w_{ij}^{[l]} x_i^{[l-1]}}} \end{aligned} \tag{4.4}$$

Ähnlich wie bei der [0,1]–Einschränkung ergibt sich:

$$\tilde{\mu} = N\mu \tag{4.5}$$

Die Gleichung 4.5 deutet darauf hin, daß die Nebenwirkung der mittelnden Additionen auf die Aktivität des Neurons durch eine steilere Sigmoidfunktion ausgeglichen werden kann. Je größer die Anzahl der Neuroneneingänge ist, desto steiler sollte die Sigmoidfunktion sein. Für das XOR–Problem ist die Auswirkung des $[\frac{1}{N}]$–Problems allein sehr gering, weil die Neuronen im Netz höchstens drei Eingänge haben. Die Simulationsergebnisse

werden in Abbildung 4.3 gezeigt. Außer der Steilheit μ sind alle anderen Lernparameter in der Simulation für die beiden Fälle (ohne und mit $\frac{1}{N}$–Mittelungsverfahren) mit gleichen Werten gewählt. Aus dieser Abbildung ist deutlich zu sehen, daß der Verlauf des Lernfehlers im Fall des $\frac{1}{N}$–Mittelungsverfahrens mit $\mu = 3$ fast identisch ist zu dem originalen Fall mit $\mu = 2$. So dürfte diese Einschränkung allein keine Auswirkung auf die Konvergenzeigenschaft des Verfahrens haben. Sie kann lediglich die Auswahl der Lernparameter und die Geschwindigkeit des Trainings leicht beeinflussen.

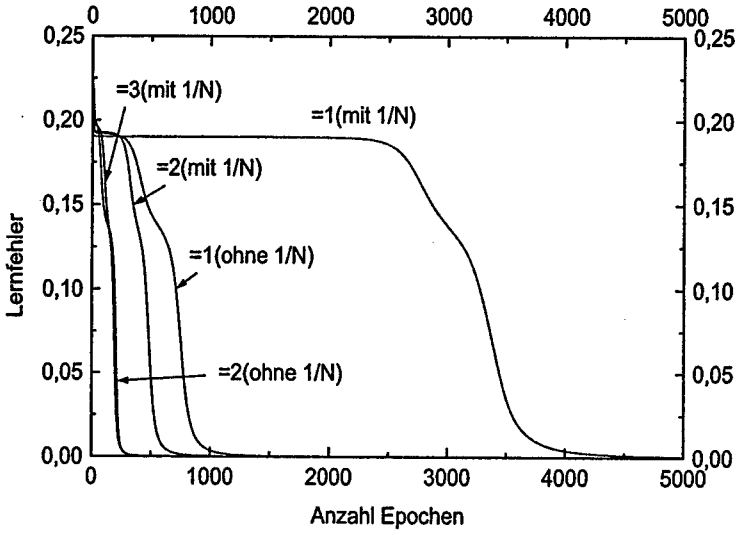


Abb. 4.3: Auswirkung des $\frac{1}{N}$–Mittelungsverfahrens beim XOR–Problem

4.2.1.3 Squash– und Bogenfunktion

Die Verwendung einer Sigmoidfunktion ist im BP–Algorithmus kein Muß. Das Gradientenabstiegsverfahren fordert nur, daß die Überführungsfunktion der Neuronen streng monoton steigend ist und sich asymptotisch den Grenzwerten nähert. Offensichtlich erfüllt die Squash– bzw. S–Funktion solche Bedingungen. So kann sie die Sigmoidfunktion im BP–Algorithmus ersetzen. Aber die ideale Ableitung der S–Funktion, welche für die Konvergenz des Verfahrens eine entscheidende Rolle spielt, ist aus Sicht der Hardware nicht leicht zu implementieren. Statt dessen wird eine Bogenfunktion (B–Funktion) verwendet, welche die ideale Ableitung der S–Funktion annähert und in Hardware leicht implementiert werden kann. In [51] wurde der

Konvergenznachweis bisher nur unter der Annahme idealer Ableitungen der Squashfunktion erbracht. Nun wird bewiesen, daß das Training trotz des Einsatzes der B–Funktion als Ersatz der idealen Ableitung der S–Funktion weiterhin konvergieren kann.
Sei Σ die Anzahl sämtlicher Gewichte und $\vec{W}$ ein langer Vektor der Dimension $\Sigma(\vec{W} \in R^{\Sigma})$, welcher alle Gewichte des Netzes umfaßt. Kurz geschrieben, $\vec{W} = (w_{ij})$. Die Aufgabe des BP-Algorithmus entspricht der folgenden Problemstellung einer Optimierung: Suche nach dem entsprechenden $\vec{W}^*$, so daß

$$E(\vec{W}^*) = min. \tag{4.6}$$

E(...) ist die nichtlineare Gesamtfehlerfunktion. Das Problem wird mit dem BP-Algorithmus iterativ gelöst. Im allgemeinen läuft das Verfahren wie folgt ab:
Gegeben sei der initiale Gewichtsvektor $\vec{W}_0$. Eine Folge $\vec{W}_k$ werde iterativ so erzeugt, daß $\vec{W}_k$ gegen $\vec{W}^*$ konvergiert:

$$\vec{W}_{k+1} = \vec{W}_k + \lambda_k \vec{S}_k \tag{4.7}$$

λ_k wird als Suchschrittweite bezeichnet und $\vec{S}_k$ als Suchrichtung. Für das Anlegen des p–ten Trainingsmusters beim Eingang wird die Suchrichtung als $\vec{S}_k(p)$ bezeichnet. In [22] wurde bewiesen:
Wenn die Bedingung

$$(\vec{S}_k)^T \nabla E(\vec{W}_k) < 0 \tag{4.8}$$

erfüllt wird, dann existiert eine Konstante $\bar{\lambda} > 0$, so daß für alle λ mit $0 < \lambda \leq \bar{\lambda}$ gilt:

$$E(\vec{W}_{k+1}) := E(\vec{W}_k + \lambda \vec{S}_k) < E(\vec{W}_k) \tag{4.9}$$

Die Ungleichung(4.9) in Verbindung mit $E(\vec{W}) \geq 0$ gewährleistet, daß die Folge $\{\vec{W}_k\}$ gegen $\vec{W}^*$ oder gegen ein lokales Minimum konvergiert.
Im herkömmlichen BP-Algorithmus wird der negative Gradient der Fehlerfunktion $-\nabla E(\vec{W}_k)$ als Suchrichtung genommen, daher ist (4.8) klar erfüllt. Der Gradient im BP läßt sich so ermitteln [57]:

$$\begin{aligned} \nabla E_p(\vec{W}_k) &= \left(\frac{\partial E_p}{\partial w_{11}(k)}, \frac{\partial E_p}{\partial w_{12}(k)}, \ldots, \frac{\partial E_p}{\partial w_{ij}(k)}, \ldots\right)^T \\ \frac{\partial E_p}{\partial w_{ij}(k)} &= \delta_{pi} O_{pj} \\ &= O_{pj} f_i' \sum_l \delta_{pl} w_{li}(k) \end{aligned} \tag{4.10}$$

mit

p : Muster-Index

k : Iterationsindex

i, j : Neuron-Index

l : Summationsindex

O_{pj} : Ausgangssignal des j–ten Neurons beim Anlegen des p–ten Trainingsmusters beim Netzeingang

f_i' : Ableitung der Sigmoidfunktion des i–ten Neurons

Wenn die B–Funktion zum Einsatz kommt, so läßt sich die Suchrichtung $\vec{S}_k(p)$ folgendermaßen ermitteln:

$$\vec{S}_k(p) \stackrel{def}{=} (s_{ij}^k(p)) \tag{4.11}$$

$$s_{ij}^k(p) = -O_{pj} b_i \sum_l \delta_{pl} w_{li}(k) \tag{4.12}$$

Dabei ist b_i die Bogenfunktion des i–ten Neurons.
Die Squashfunktion verläuft im Maschinen-Bereich [-1,1] streng monoton steigend. Demnach ist ihre ideale Ableitung f' in dem Bereich stets positiv. Die B–Funktion ist im Bereich (-1,1) ebenfalls immer positiv. Dadurch ergibt sich:

$$s_{ij}^k(p) * \frac{\partial E_p}{\partial w_{ij}(k)} = -O_{pj}^2 (f_i' b_i)(\sum_l \delta_{pl} w_{li}(k))^2 < 0 \tag{4.13}$$

So ist die Abstiegsbedingung (Ungleichung 4.8) sofort erfüllt, wenn nicht alle Gewichte gleich Null sind. Damit ist die Konvergenz des Verfahrens nachgewiesen. Zwar ist die Suchrichtung nicht mehr die Richtung des lokal steilsten Abstiegs der Fehlerfunktion, aber doch so, daß der Lernprozeß konvergent verläuft.
In der Tat bringt der Einsatz der S– und B–Funktionen kaum Schwierigkeiten beim Training eines konventionellen BP–Netzes. In Abschnitt 2.3.3 wurde bereits erläutert, daß die S–Funktion nur im Intervall $[-1, 1]$ eine gute Anpassung an die Sigmoidfunktion besitzt. Dies bedeutet, daß die Eingangssignale normierte Daten sein sollten. Um dies zu garantieren, ist von der $[0, 1]$–Einschränkung die Rede. Die Abbildung 4.4 zeigt einige Ergebnisse von Simulationsläufen, welche mit unterschiedlichen Parametern für das XOR–Problem durchgeführt wurden.

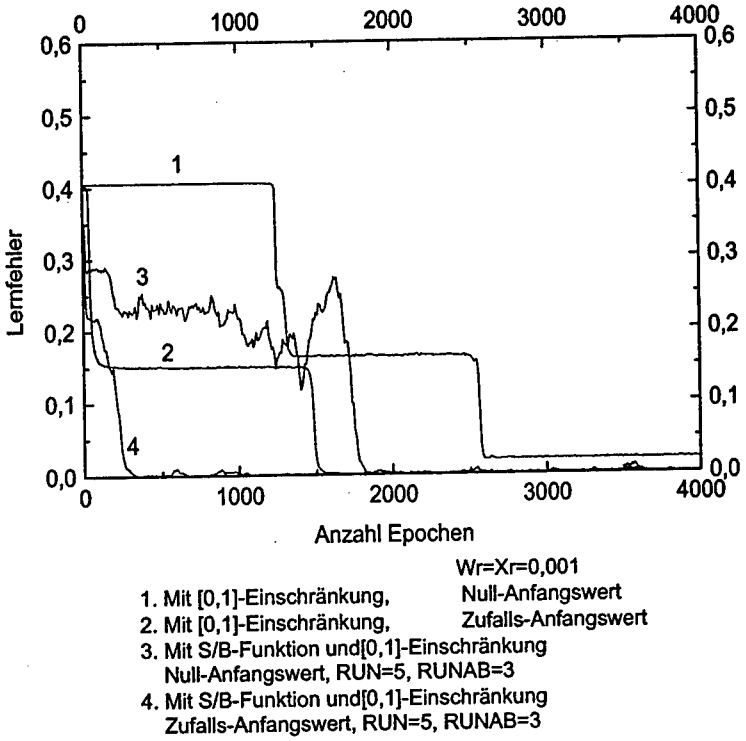


Abb. 4.4: *Auswirkung des Einsatzes der S– und B–Funktion bei der Lösung des XOR–Problems*

Es ist in der Abbildung 4.4 deutlich zu sehen, daß das Training beim Einsatz der S– und B–Funktionen schneller konvergiert als im Fall der traditionellen Sigmoidfunktion. Diese Ergebnisse zeigen nicht nur eine Bestätigung der obigen theoretischen Herleitung, sondern auch die vorteilhafte Eigenschaft des Einsatzes der S– und der B–Funktion gegenüber dem Einsatz der steilen Sigmoidfunktion, wenn starkes Rauschen und $[0,1]$–Einschränkung beim Training auftreten. Dies ist genau beim hier erläuterten Verfahren der Fall. In der Simulation ergeben sich beim Einsatz der Sigmoidfunktion mit der Steilheit $\mu = 8$, welche der Steilheit der S–Funktion mit $RUN = 5$ entspricht, große Konvergenz–Schwierigkeiten, obwohl ihre echte Ableitung berechnet wurde und somit die lokal steilste Abstiegsrichtung zur Wirkung kam. Der Grund liegt darin, daß die Sigmoidfunktion wegen ihrer Steilheit sehr oft in den Sättigungsbereich gerät und ihre Ableitung

$s'(x) = \mu s(x)[1 - s(x)]$ dort sehr leicht ein umgekehrtes Vorzeichen haben kann, wenn bei ihrer Berechnung starkes Rauschen eine Rolle spielt. So könnte die dadurch bestimmte Suchrichtung genau entgegengesetzt sein zur Richtung des lokal steilsten Abstiegs der Fehlerfunktion. Demzufolge wird das Training mehr Epochen benötigen als ohne Rauschen. Dagegen hat die B–Funktion nichts zu tun mit der Steilheit beziehungsweise mit dem Sättigungsbereich der S–Funktion.

4.2.1.4 Zusammenwirken der Einschränkungen

In den obigen Abschnitten wurde der Einfluß der Einschränkungen einzeln untersucht. Jedoch treten alle Einschränkungen im hier vorgestellten Verfahren gleichzeitig auf. So ist es notwendig, ihr Zusammenwirken zu untersuchen und mögliche negative Auswirkungen einzuschätzen.

Die Analyse in den vorherigen Abschnitten zeigt, daß es aus theoretischer Sicht entsprechende Maßnahmen gibt, um die negativen Auswirkungen jeder Einschränkung für sich zu beseitigen oder zu kompensieren. Die Maßnahmen zur Kompensation der $[0, 1]$–Einschränkung und des $\frac{1}{N}$–Mittelungsverfahrens sind gleich, nämlich die Verwendung einer steileren Sigmoidfunktion. Logischerweise ist vorzusehen, daß eine noch steilere Sigmoidfunktion verwendet werden soll, wenn beide Einschränkungen gleichzeitig auftreten. Wie in Abbildung 4.5 gezeigt wird, liefert die Simulation ein Ergebnis, das dem vorangegangenen ähnelt.

Aus der Simulation lassen sich folgende Erkenntnisse gewinnen:

- Wenn $[0, 1]$–Einschränkung und $\frac{1}{N}$–Mittelungsverfahren gleichzeitig beim Training wirksam sind, dann kann das Training nur unter Verwendung einer sehr steilen Sigmoidfunktion zur Konvergenz kommen.
- Die Lernrate γ muß sehr klein gehalten werden, wenn die Sigmoidfunktion sehr steil ist. Andernfalls gerät das Training in die Divergenz oder bleibt in einem lokalen Minimum hängen.
- Wegen der kleinen Lernrate kann die Stärke des Rauschens eine entscheidende Rolle für die Konvergenz des Trainings spielen (siehe Fall 4 in der Abbildung 4.5).
- Außer den Gewichten zu den Bias–Eingängen liegen alle Gewichte in der Nähe ihrer zugelassenen Grenzwerte, nämlich -1 oder $+1$, wenn der Lernfehler beim Training irgendwo steckenbleibt.

Je mehr Einschränkungen gleichzeitig auftreten, umso schwieriger wird es, das entsprechende Netz erfolgreich zu trainieren. Die Abbildung 4.6 gibt eine qualitative Darstellung für die Schwierigkeit des Trainings, wenn eine,

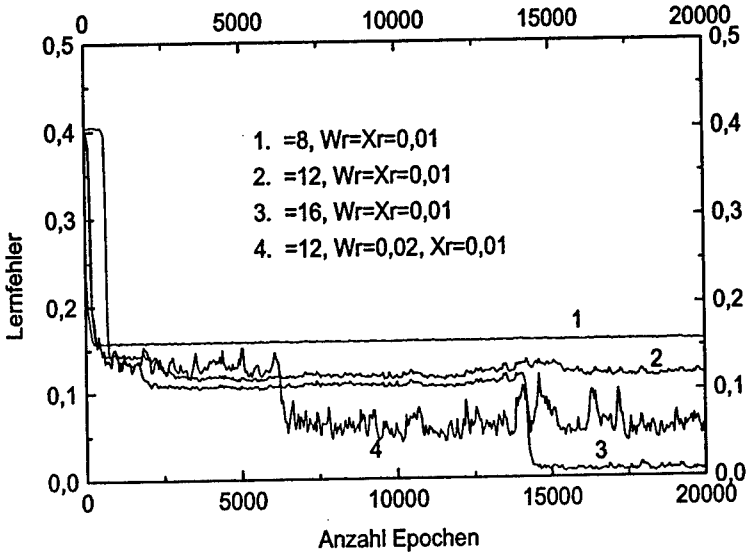


Abb. 4.5: Zusammenwirken der Einschränkungen beim XOR–Problem

zwei und drei Einschränkungen gleichzeitig im Netz auftreten. Jeder Kreis mit Text repräsentiert das Auftreten der entsprechenden Einschränkung im BP–Netz. Die Graustufen stellen das Schwierigkeitsniveau des Trainings dar (je dunkler, umso schwieriger). Am schwierigsten ist das gleichzeitige Auftreten der $[0,1]$–Einschränkung, des $\frac{1}{N}$–Mittelungsverfahrens und des Einsatzes der S– und B–Funktionen. In diesem Fall ist es sehr schwer, das entsprechende Netz mit konventioneller Rechentechnik zur Konvergenz zu bringen. Die meisten Versuche führten zu einem divergenten Trainingsverhalten.

4.2.2 BP–Netz in stochastischer Rechentechnik

Für die Realisierung von BP–Netzen mit stochastischen Rechenwerken ist die Erzeugung von Zufallsfolgen erforderlich. Aber es ist kaum möglich, eine echte Zufallsfolge (Bernoulli–Folge) zu erzeugen. Stattdessen kommen binäre Pseudozufallsfolgen (BPZF) in Betracht. Eine bekannte Familie von BPZF ist die sogenannte m–Sequenz [36], welche sich durch Modulo–2 rückgekoppelte Schieberegister sehr einfach erzeugen läßt. Im vorgestellten Ver-

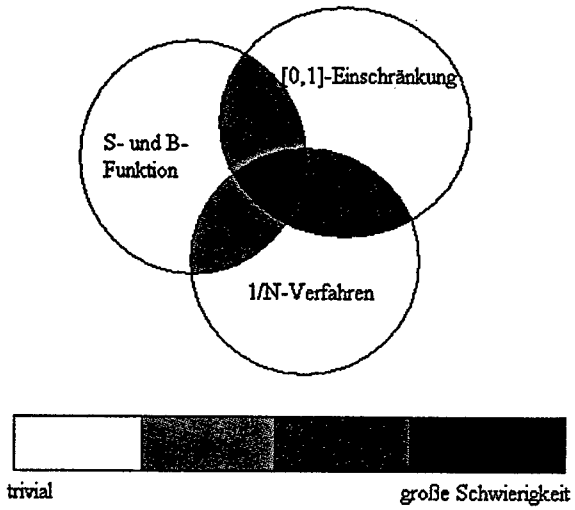


Abb. 4.6: *Schwierigkeitsgrad des Trainings in Abhängigkeit des Auftretens der einzelnen Einschränkungen*

fahren werden viele m–Sequenzen erzeugt und zum Einsatz gebracht [51]. Man darf jedoch nicht außer Acht lassen, daß m–Sequenzen nur einige und nicht alle Merkmale einer echten Zufallsfolge aufzeigen. Insbesondere treten leicht unübersichtliche Verhältnisse in Bezug auf die gegenseitige Unabhängigkeit bei gleichzeitiger Verarbeitung mehrerer m–Sequenzen in stochastischen Rechenwerken auf. Nach Massen [36] sollten m–Sequenzen in der stochastischen Rechentechnik nur dort eingesetzt werden, wo die stochastische Codierung und Decodierung im Vordergrund steht und nur einige einfache arithmetische Operationen durchgeführt werden. Dies ist in dem hier erläuterten Verfahren der Fall, weil die erzeugten m–Sequenzen meistens als Hilfsfolgen für die Codierung verwendet werden. Eine m–Sequenz hat folgende Eigenschaften, die etwa den Merkmalen einer echten Zufallsfolge entsprechen:

- Die Häufigkeit des Erscheinens einer Eins ist etwa gleich der Häufigkeit des Erscheinens einer Null in der binären Folge. Aus diesem Grund wird sie als **0,5-Folge** bezeichnet.

- Folgen von konsekutiven gleichen Ereignissen, sogenannte RUNs, sind um so häufiger, je kürzer die RUN–Länge ist. Im allgemeinen sollen etwa die Hälfte der RUNs die Länge 1 haben, ein Viertel die Länge 2 usw., d.h. die Anzahl $Z(l)$ der RUNs mit Länge l aus einer m-Sequenz mit der Länge L^* beträgt etwa:

$$Z(l) = \frac{L^*}{2^{(l+2)}} \tag{4.14}$$

Die Werte von $Z(l)$ entsprechen den unterschiedlichen Längen der RUNs und bilden eine Runlängen–Verteilung, auf welche in den folgenden Abschnitten Bezug genommen wird. Abbildung 4.7 zeigt die Runlängen–Verteilung einer m-Sequenz, welche mit der in [51] erläuterten Technik erzeugt wird. Die Runlängen–Verteilung stimmt mit der Gleichung 4.14 überein.

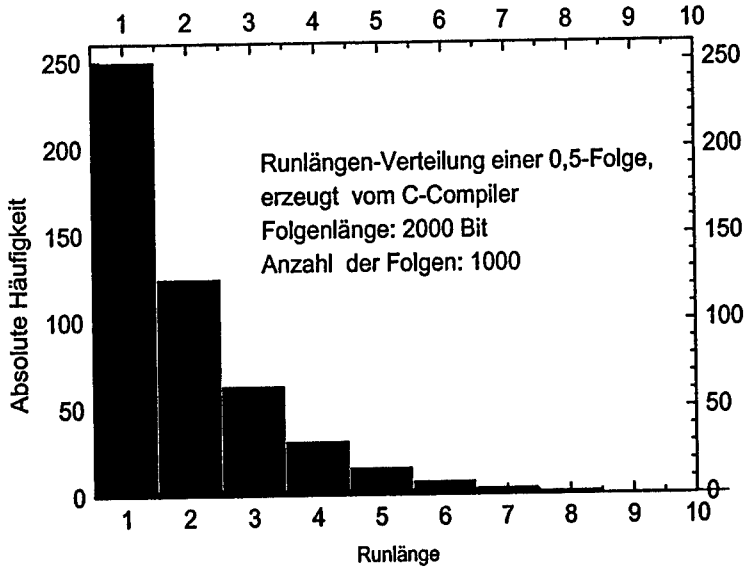


Abb. 4.7: Runlängen–Verteilung einer 0,5-Folge

4.2.2.1 Einsatz von S– und B–Funktion

Wie im Abschnitt 2.3.3 erläutert wurde, werden die S– und die B–Funktion durch Automaten implementiert. Die Automaten werden dementsprechend als S– und B–Automat bezeichnet und entsprechen einfachen Runlängen–Akzeptoren. So spielt die Runlängencharakteristik der Eingangsfolgen für

die Reaktion des Automaten eine große Rolle. In einem MLN sind Neuronenschichten seriell miteinander verbunden. Das bedeutet, daß die Ausgangsfolge eines Neurons der dahinter liegenden Schicht als Eingangsfolge zugeführt wird. Daher ist es notwendig, die Runlängencharakteristik der Ausgangsfolge eines S–Automaten zu studieren, wenn dieser für die Implementierung der S–Funktion zum Einsatz kommt. Für die Untersuchung wird folgendes Schema verwendet: Ein deterministischer Wert $P = 0,5$ wird zuerst unter Hinzunahme einer Hilfsfolge (einer m-Sequenz) in eine binäre Folge codiert und dann in den S–Automaten eingeführt. Außerdem werden zwei Monitore (M1 und M2) eingerichtet, um die Runlängen–Verteilung der entsprechenden binären Folgen zu beobachten (siehe Abbildung 4.8).

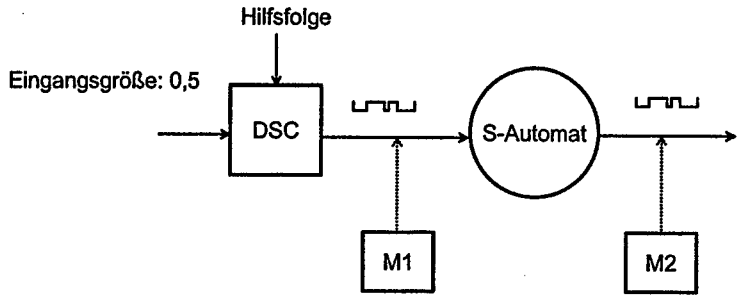


Abb. 4.8: Blockschema für die Untersuchung der Runlängencharakteristik eines S–Automaten

Die in M1 beobachtete binäre Folge, die sogenannte Eingangsfolge des S–Automaten, hat dieselbe Runlängencharakteristik wie eine m-Sequenz. Ihre Runlängen–Verteilung wurde bereits in der Abbildung 4.7 gezeigt. Dagegen entspricht die Runlängen–Verteilung der Ausgangsfolge des S–Automaten (beobachtet durch M2) nicht mehr einer m-Sequenz. Die Folge besteht aus nur wenigen RUNs, die sehr lang sind (siehe Abbildung 4.9). Je steiler die S–Funktion ist, umso geringer ist die Anzahl der RUNs und umso längere RUNs besitzt ihre Ausgangsfolge, obwohl die Häufigkeit des Erscheinens einer Eins dennoch um 0,5 liegt.

Nun entspricht die Ausgangsfolge nicht mehr einer m-Sequenz, sondern ist eine Folge, deren Verhalten durch das Bildungsgesetz des S–Automaten geprägt ist. Diese binäre Folge verliert damit von jetzt an ihre ursprüngliche stochastische Unabhängigkeit und RUN–Längenverteilung, welche aber eine Voraussetzung für die weiteren Verarbeitungsschritte ist. Wird diese Ausgangsfolge einem S–Automaten in der nachfolgenden Schicht zugeführt, funktioniert letzterer quasi nur wie eine lineare Funktion. Glücklicherweise

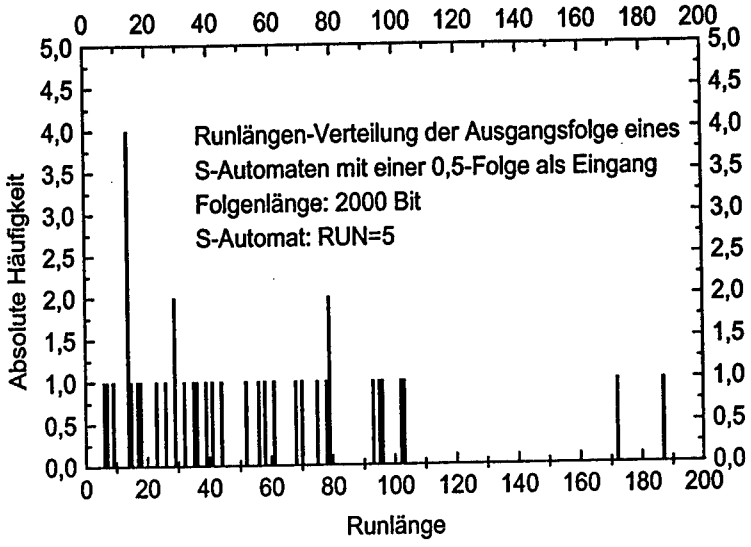


Abb. 4.9: Runlängen–Verteilung der Ausgangsfolge eines S–Automaten

findet noch eine weitere Operation dazwischen statt. Dies ist die Multiplikation der Ausgangsfolge des vorherigen S–Automaten mit dem Gewicht w, welches durch eine 0,5–Folge (eine m-Sequenz) in eine binäre Folge codiert wird. Dadurch kann die erwartete Runlängencharakteristik und damit auch die stochastische Unabhängigkeit wieder hergestellt werden, wenn das Gewicht den Wert Null (im M–Bereich) annimmt oder in der Nähe von Null liegt. Die Abbildung 4.10 zeigt die Runlängen–Verteilung der Folge, die nach der Multiplikation zwischen dem Neuronenausgang und einem Gewicht $w = 0$ entsteht. Es ist deutlich zu sehen, daß die typische Runlängencharakteristik einer m-Sequenz wieder hergestellt ist.

Je weiter allerdings das Gewicht von Null entfernt ist, desto weniger kann diese Gewichtung die verlangte Verteilung wieder herstellen. Nimmt das Gewicht einen maximalen Betragswert (-1 oder 1) an, ist die Wiederherstellung der Verteilung nicht mehr möglich (siehe Abbildung 4.11). Daher hat das nachfolgende Neuron (implementiert durch einen S–Automaten) nicht mehr die ihm zugedachte Wirkung.

Wegen der Überlaufsperre des Gewichtsglieds muß man bevorzugt von einem maximalen Betragswert eines Gewichts beim Training ausgehen. Denn die Ausgangsfolge eines B–Automaten, die zur Gewichtskorrektur verwendet werden soll, ist auch keine m-Sequenz mehr und besteht ebenfalls aus eini-

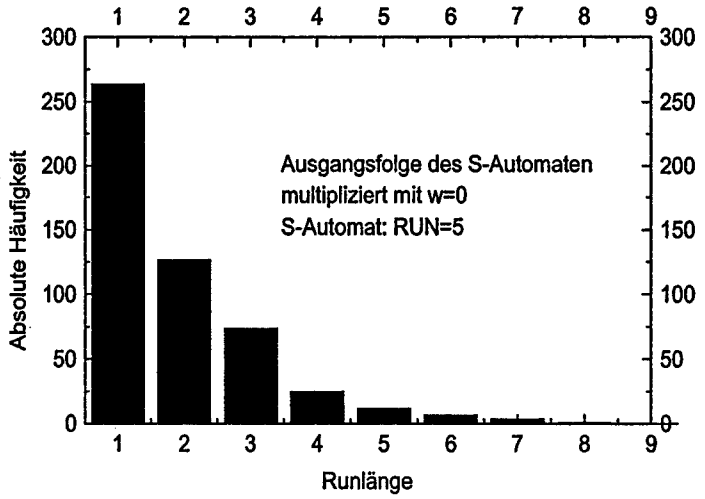


Abb. 4.10: Runlängen–Verteilung der nach der Multiplikation zwischen dem Neuronenausgang und einem Gewicht $w = 0$ entstehenden Folge

gen langen RUNs. Ein langes RUN kann den Zähler in einem INDIE–Glied zum Überlauf bringen. Dadurch kann die oben genannte ungünstige Situation während des Trainings häufig eintreten. Nehmen jedoch alle Gewichte in einem MLN die maximalen Betragswerte an und zeigen die Eingänge aller Neuronen in der ersten Schicht eine 0,5–Folge, dann bleibt das Training stecken. Nun stellt sich die Frage, wie die Eingangsfolge eines Neurons eine 0,5–Folge werden kann, wenn nur Einsen und Nullen am Eingang des Netzes angelegt werden. Zurückblickend auf die Arbeitsweise des Netzes mit stochastischen Rechenwerken werden die Eingangswerte in binäre Folgen codiert und dann mit entsprechenden Gewichten multipliziert, welche während des Trainings dann leicht extreme Werte erreichen können. Danach wird eine Eingangsfolge für jedes Neuron durch eine mittelnde Addition (implementiert durch einen Multiplexer) erzeugt (siehe Abbildung 4.12).

Die Gewichtswerte liegen im Intervall $[0, 1]$, wenn sie im W–Bereich betrachtet werden, und bewegen sich während des Trainings in diesem Intervall gemäß dem Lernalgorithmus oder auch zufällig, je nachdem wie stark die stochastische Streuung ist. In diesem Sinn könnte jedes Gewicht als eine in dem Intervall $[0, 1]$ gleichverteilte stochastische Variable betrachtet werden. Infolgedessen wird der Mittelwert aller Gewichte 0,5 sein, wenn die Anzahl

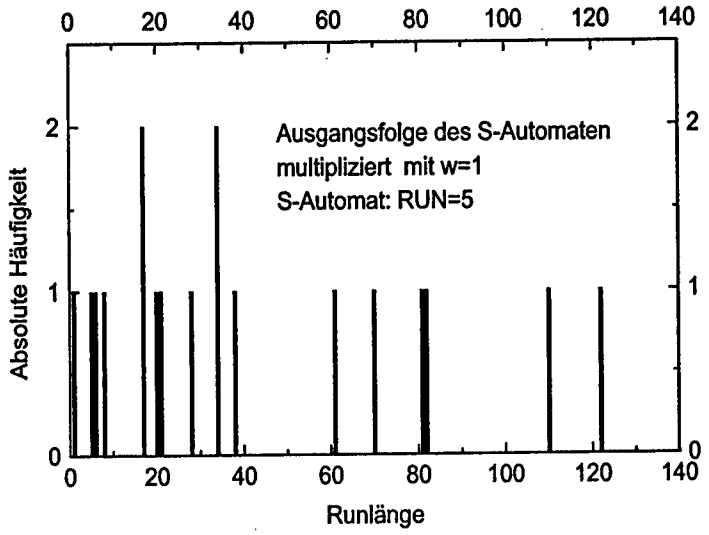


Abb. 4.11: Runlängen–Verteilung der nach Multiplikation zwischen dem Neuronenausgang und einem Gewicht $w = 1$ entstehenden Folge

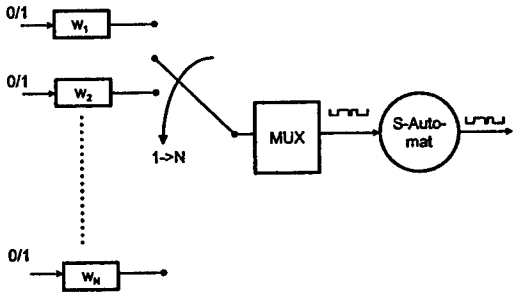


Abb. 4.12: Blockschema zur Erzeugung der Eingangsfolge eines Neurons

N der Gewichte vor einem Neuron groß genug ist. Dadurch wird die Resultatfolge der mittelnden Addition, welche als Eingangsfolge des nachstehenden Neurons dient, sicher eine 0,5–Folge sein. Für diesen Ansatz sollen weitere theoretische Analysen oder Simulationen durchgeführt werden.

Noch eine weitere Eigenschaft des S–Automaten ist während der Simulation aufgefallen. Der Verlauf der S–Funktion, die der S–Automat liefern soll, wird sehr stark von der Verteilung der binären Werte seiner Eingangsfolge beeinflußt. Diese Eigenschaft entstammt der Arbeitsweise des S–Automaten, der lediglich einen Runlängen–Akzeptor darstellt. Das heißt, daß die S–Funktion bei unterschiedlichen DSCs (siehe Abbildung 4.8) ganz andere Verläufe zeigen kann, obwohl die zu verarbeitenden Eingangsfolgen gleichen Werten im M–Bereich entsprechen. Die Abbildung 4.13 zeigt den Verlauf der S–Funktion bei einem traditionellen komparatorischen Codierer, während die Abbildung 4.14 den Verlauf bei Verwendung eines technisch vorteilhafteren Codierers zeigt. Damit verhält sich der Automat in beiden Fällen sehr unterschiedlich. Im ersten Fall ist der Verlauf fast wie erwartet. Demgegenüber sieht der Verlauf im letzten Fall fast wie eine Stufenfunktion aus. Dieses Verhalten könnte beim Training des Netzes zu unerwarteten Folgen führen, z.B. zur Divergenz des Trainings. In den Speichergliedern (ADDIE und INDIE) werden sequentielle Codierer verwendet, deren Ausgang (codierte binäre Folge) danach direkt einem S–Automat zugeführt wird. Aufgrund der starken Streuung in der Nähe des Ursprungs kann das Ausgangssignal des S–Automaten total verfälscht werden. Damit wird die Suchrichtung völlig falsch berechnet.

Theoretisch soll der S–Automat eine 0,5–Folge liefern, wenn ihm eine 0,5–Folge zugeführt wird. Aber eine Abweichung ist in der Tat immer festzustellen, weil bei der Verarbeitung mit stochastischen Rechenwerken überall stochastische Streuungen bestehen. Es soll deshalb weiter untersucht werden, ob die ursprüngliche stochastische Streuung durch die Nichtlinearitäten, d.h. die S– und B–Automaten, weiter vergrößert wird. Dies wird im nächsten Abschnitt untersucht.

4.2.2.2 Ausbreitung der stochastischen Streuung

Das Einbringen von Rauschen in den Lernalgorithmus ist eine bewährte Technik, um beim konventionellen Algorithmus zur Verbesserung der Konvergenz beizutragen. Im vorgestellten Verfahren ist die Existenz der stochastischen Streuung systemimmanent und beruht auf der speziellen Implementierung. Nach der Erläuterung im vorigen Abschnitt kann die Stärke der stochastischen Streuung beim gleichzeitigen Auftreten beider Einschränkungen, d.h. der $[0,1]$–Einschränkung und des $\frac{1}{N}$–Mittelungsverfahrens, eine entscheidende Rolle für die Konvergenz des Trainings spielen. Aus diesem Grund ist es notwendig, die Stärke der Streuung im Verfahren und insbesondere ihre Ausbreitung in einem MLN zu untersuchen.

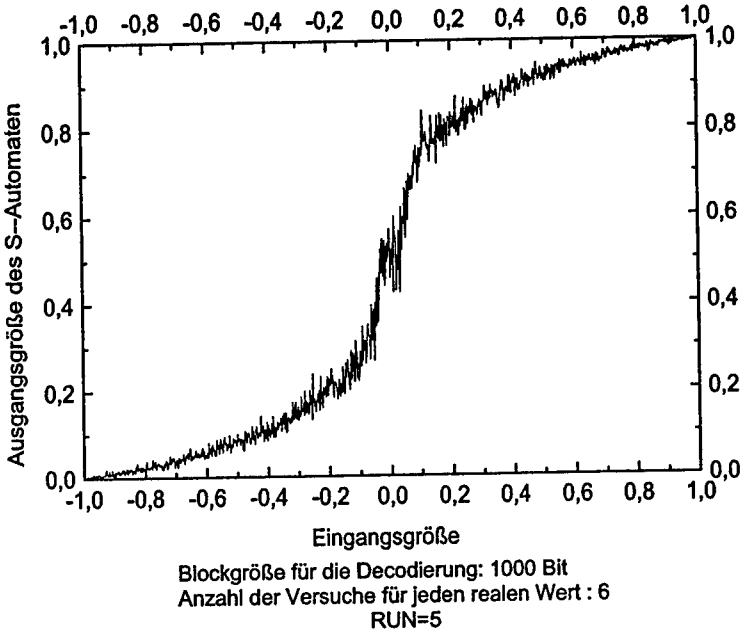


Abb. 4.13: Verlauf der S–Funktion mit einem komparatorischen Codierer

Für die Untersuchung ist ein Maß nötig, durch das die Stärke der Streuung gemessen werden kann. Dafür werden folgende Größen in der nachstehenden Analyse vereinbart:

- **Durchschnittlicher Meßfehler**
  Sei t das theoretische Resultat einer Berechnung und n die Anzahl der Versuche, bei denen Meßwerte x_j $(j = 1, ..n)$ für den theoretischen Wert t durch ein bestimmtes stochastisches Rechenwerk gewonnen werden. So beträgt der durchschnittliche Meßfehler:

$$\varepsilon = \sqrt{\frac{1}{n}\sum_{j=1}^{n}(x_j - t)^2} \tag{4.15}$$

- **Standardabweichung**
  Sei $\bar{x}$ der Mittelwert der Meßwerte x_j $(j = 1, ..n)$ aus n Versuchen, dann wird eine Standardabweichung zum Mittelwert (nach Kreyszig

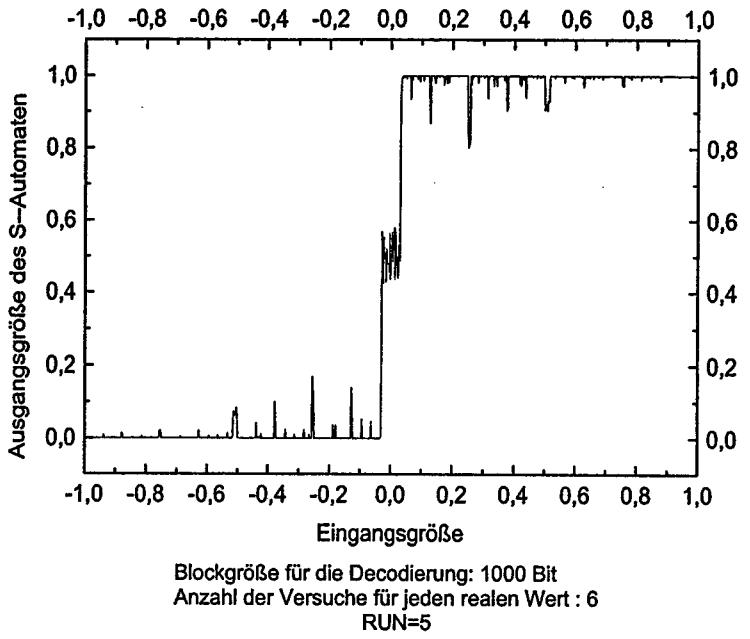


Abb. 4.14: *Verlauf der S–Funktion mit einem sequentiellen Codierer*

[29]) so definiert:

$$\sigma = \sqrt{\frac{1}{n-1}\sum_{j=1}^{n}(x_j - \bar{x})^2} \tag{4.16}$$

Offenbar ist obiges Maß für den durchschnittlichen Meßfehler nicht sonderlich für eine Nachbildung in Hardware geeignet, weil der theoretische Wert t nicht zur Verfügung steht. In einem solchen Fall ergibt aber die Standardabweichung ein gutes Maß. Für eine Software–Simulation, die eine Nachbildung des Verfahrens in Software implementiert, kann beides verwendet werden.

Für die Untersuchung wird ein minimales MLN betrachtet, das in der Abbildung 4.15 gezeigt wird. Es ist ein zweischichtiges MLN mit nur einem Signal–Eingang und einem Neuron je Schicht. Die Gewichte sind auf feste Werte gelegt und das Eingangssignal ändert sich im W–Bereich von 0,0 bis 1,0 mit einer Schrittweite von 0,002. Jeder Wert wird T Takte lang am Netzeingang angelegt und in eine binäre Folge codiert. Danach wird eine

Multiplikation mit dem festen Gewicht durchgeführt und die Resultatfolge dem Neuron (S–Automat) zugeführt, usw. Bei jedem Eingangswert wird dieser Vorgang n mal wiederholt, um ausreichend Stichproben zu erhalten. Jede Stichprobe x_j wird durch das Summieren über T Takte hergestellt. Der theoretische Wert t läßt sich durch die Gleichungen 2.5 und 2.26 aus den Abschnitten 2.2.2.2 und 2.3.3 berechnen, und zwar ohne Berücksichtigung des Rundungsfehlers. An vier Stellen wird ein Monitor M_i $(i = 1...4)$ eingerichtet, durch den die entsprechende binäre Folge beobachtet werden kann.

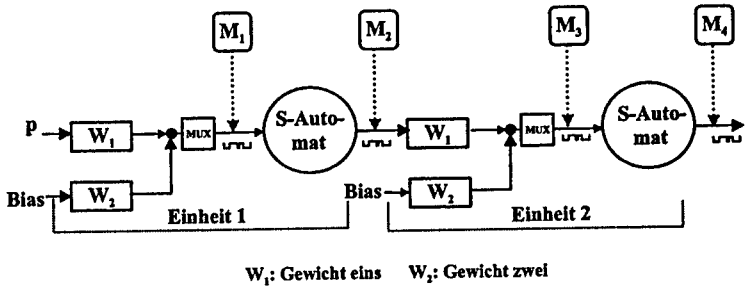


Abb. 4.15: Schematische Anordnung zur Untersuchung der Ausbreitung der stochastischen Streuung

Es werden die durchschnittlichen Meßfehler der von den Monitoren 1-4 gemäß Abbildung 4.16 beobachteten Werte berechnet und in Abbildung 4.17 im Vergleich zur Standardabweichung dargestellt. Beim ersten ist deutlich zu sehen, daß sich der durchschnittliche Meßfehler der Ausgangsfolge des S–Automaten vergrößert, wenn sich der Beobachtungspunkt vom Eingang entfernt. Insbesondere nimmt der durchschnittliche Meßfehler in der Nähe des Wertes 0,5, der dem Ursprung im M–Bereich entspricht, den größten Wert an. In diesem Bereich befindet sich gerade der steil ansteigende Teil der S–Funktion, der im Laufe des Trainings die entscheidende Rolle spielt. Aber die stochastische Streuung in dem Bereich kann bis zu 20% groß sein. Dies könnte dazu führen, daß das Training in diesem Wertebereich mehr zufällig als durch den Algorithmus selbst gesteuert wird. Je mehr Schichten eine binäre Folge durchlaufen muß, umso stärker ist der Einfluß der stochastischen Streuung und um so zufälliger verläuft das Training. Weitere Simulationen zeigen auch, daß die Erhöhung der Taktanzahl oder des Stichprobenumfangs keine nennenswerte Verbesserung für die Größenordnung der stochastischen Streuung bietet. Weiterhin wird die stochastische Streuung noch größer, wenn sich die Steilheit der S–Funktion, nämlich der

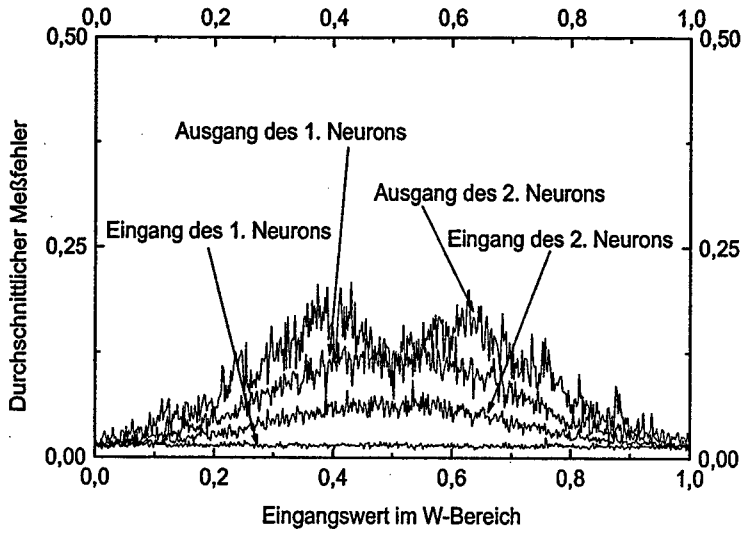


Abb. 4.16: *Ausbreitung des durchschnittlichen Meßfehlers in einem MLN mit stochastischen Rechenwerken*

Wert des Parameters RUN, erhöht (siehe Abbildung 4.18). Im zweiten Fall ist zu sehen, daß die Standardabweichung zwar nicht durch mehrere hintereinandergeschaltete S–Automaten vergrößert wird, aber sie kann noch bis zu 15% groß sein. In diesem Fall ist die Nichtlinearität des Neurons allein die Hauptursache der Vergrößerung der Standardabweichung. Logischerweise ist die Standardabweichung ein geeignetes Maß für die Untersuchung, weil nur der Mittelwert $\bar{x}$ in die Datenverarbeitung einbezogen wird. In der Tat spielt der theoretische Wert t im Training überhaupt keine Rolle. Der beim Netzausgang für den Vergleich zwischen den Ziel– und Netzausgaben verwendete Wert ist der Mittelwert, der durch Summierung der Einsen aus der entsprechenden Bitfolge gewonnen wird. Wenn nur die Standardabweichung in die Betrachtung miteinbezogen wird, kann die Streuung bei steilerer S–Funktion doch noch sehr groß werden (siehe Abbildung 4.19).

In den obigen Abbildungen werden Gewichte w_1 und w_2 zur besseren Visualisierung der Ergebnisse für beide Einheiten auf die Werte 1,0 und 0,5 festgelegt. Dabei zeigt sich, daß die Standardabweichung dann am größ-

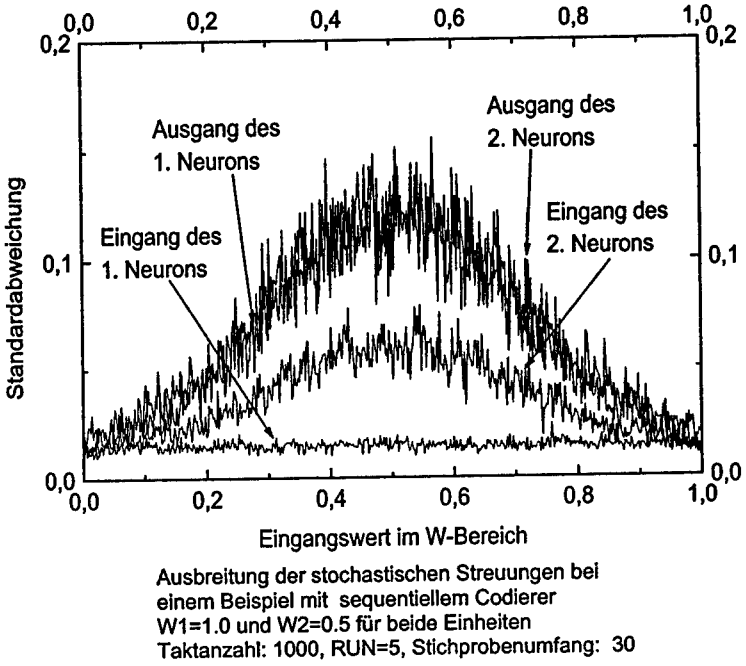


Abb. 4.17: Ausbreitung der Standardabweichung in einem MLN mit stochastischen Rechenwerken

ten ist, wenn der Eingang eines S–Automaten eine 0,5–Folge ist. Wenn die Gewichte auf andere Werte festgelegt werden, ist die Standardabweichung dann am größten, wenn der Term $\frac{(2w_1-1)(2x-1)+2w_2-1}{2}$ gleich 0,0 (bei der Betrachtung im M–Bereich) ist.

In den vorherigen Abschnitten wurde bereits festgestellt, daß das gleichzeitige Auftreten der $[0,1]$–Einschränkung und des $\frac{1}{N}$–Mittelungsverfahrens eine stark negative Auswirkung auf die Konvergenz des Trainings ausüben kann und daß dies nur durch eine steilere Aktivierungsfunktion der Neuronen kompensiert werden kann. Aus den Ergebnissen dieses Abschnitts wird die Erkenntnis gewonnen, daß bei steilerer S–Funktion die Ausgabe des Neurons sehr starke Streuungen enthalten kann, die das Training unter gewissen Umständen zur Divergenz bringen können, weil dann das Training allein durch stochastische Streuung kontrolliert wird. Dies führt zu einander widersprechenden Forderungen: Für die Kompensation der Einschränkungen wird einerseits eine steilere S–Funktion gefordert. Anderseits ist die stochastische

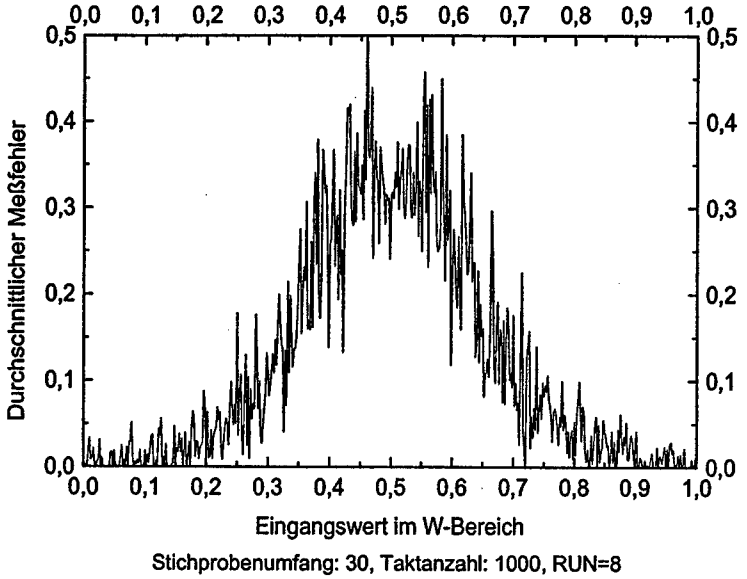


Abb. 4.18: *Ausbreitung des durchschnittlichen Fehlers in einem MLN mit einer sehr steilen S–Funktion*

Streuung um so stärker und die Runlängenverteilung der Ausgangsfolge um so schlechter, je steiler die S–Funktion ist. Dieser Widerspruch könnte zu einem großen Hindernis werden, wenn das Verfahren in größeren Netzen zum Einsatz kommen soll. Denn die Anzahl der Eingänge jedes Neurons wächst einerseits mit der Netzgröße, andererseits werden dann die negativen Auswirkungen des $\frac{1}{N}$–Mittelungsverfahrens ebenfalls größer. Das führt zu einer noch steileren Aktivierungsfunktion. Wird jeder über die zu einem Neuron führende Verbindung übertragene Wert X_i als eine im Bereich $[0,1]$ liegende Zufallsvariable betrachtet, so liefert der Ausgang des Multiplexers den Stichprobenmittelwert $\bar{x}$ der Zufallsvariablen $\bar{X} = \frac{(X_1+\ldots+X_N)}{N}$. Aus statistischer Sicht wird sich dieser Wert $\bar{x}$ dem Wert 0,5 nähern, wenn N groß wird ($N \geq 30$ [29]). Das bedeutet, daß die Eingangsfolge jedes Neurons eine quasi 0,5–Folge sein wird, wenn nur N groß genug ist.

4.2.2.3 Gegenstrom–Verfahren

Bei der konventionellen Methode wird das Training in zwei Phasen geteilt, nämlich die sogenannte Arbeitsphase und die Lernphase. In der Arbeitsphase wird die Neuronenausgangsaktivität durch Anlegen eines Eingangs-

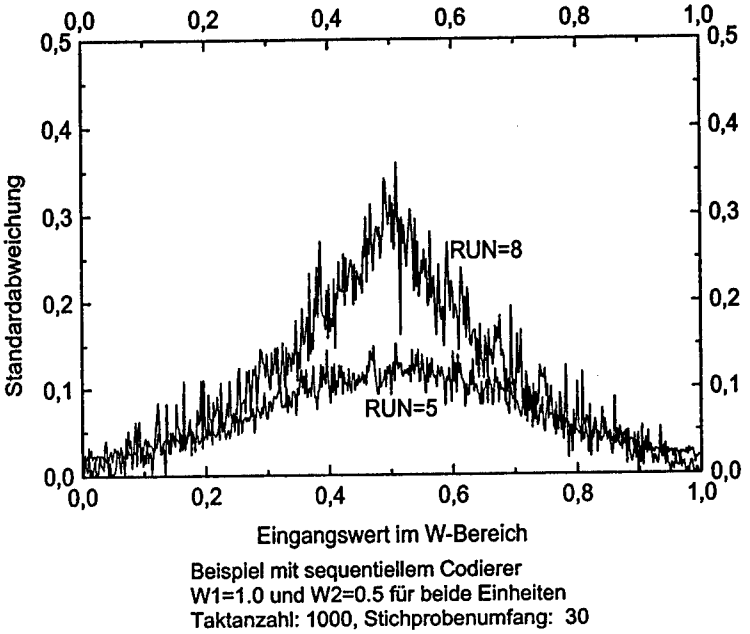


Abb. 4.19: Ausbreitung der Standardabweichung bei unterschiedlicher Steilheit des S–Automaten

vektors aus dem Trainingsmustervorrat hergestellt, während eine Gewichtsmodifikation durch Zurückgreifen der Neuronenausgangsaktivität aus der vorangegangenen Arbeitsphase nur in der Lernphase stattfindet. Daher sind zahlreiche Zwischenwerte zu speichern, die selbstverständlich zusätzlichen Speicherbedarf erfordern. Aus diesem Grund wurde in [51] ein sogenanntes Gegenstrom–Verfahren angewendet, bei dem Arbeits– und Lernphase gleichzeitig stattfinden. Auf diese Weise ist das Speichern der Zwischenwerte nicht mehr nötig; andererseits entspricht das Training nicht mehr der aus der Literatur über neuronale Netze bekannten Online–Variante des Backpropagation–Verfahrens. Ob eine solche Abweichung die Konvergenz des Netzes beim Training stark beeinflussen kann, soll intensiv untersucht werden.

Im Gegensatz zu anderen Einschränkungen, die getrennt von allen anderen Faktoren allein untersucht werden, kann die Auswirkung des Gegenstrom–Verfahren nur beim Betrieb des ganzen Netzes beobachtet werden. Um deren Auswirkung auf die Konvergenz des Trainings zu studieren, wäre es die

beste Methode, die Trainingsergebnisse vom Gegenstrom–Verfahren und der bekannten Online–Variante des Backpropagation–Verfahrens zu vergleichen. Aber man ist zu diesem Zeitpunkt noch nicht in der Lage, das vorhandene Netz, das auf Bitstrom basiert, mit einer Online–Lernstrategie zu betreiben, weil die vorhandene Architektur des Neurons nur für das Gegenstrom–Verfahren geeignet ist. Um dieses Vorhaben zu erreichen, ist ein Umbau des Neurons nötig. Aus diesem Grund wird die wesentliche Untersuchung und Diskussion in der Tat erst in den kommenden Abschnitten durchgeführt, weil die Modifikation der Struktur des Neurons und der speichernden Glieder als Gegenmaßnahmen erst dort vorgenommen wird.

4.3 Gegenmaßnahmen

Die Analysen in den vorherigen Abschnitten haben gezeigt, daß die $[0,1]$–Einschränkung und das $\frac{1}{N}$–Mittelungsverfahren einerseits die Konvergenz des Trainings zwar stark beeinflussen, andererseits aber mit dem vorgestellten Verfahren fest verbunden sind. Zur Bekämpfung der daraus resultierenden negativen Einflüsse ist eine sehr steile Aktivierungsfunktion erforderlich. Die für das vorgestellte Verfahren entwickelten Neuronen können diese Forderung zwar erfüllen, jedoch entstehen dadurch bei größeren Netzen große stochastische Streuungen und schlechte Runlängencharakteristiken der Ausgangsfolgen. Unter diesen Umständen gibt es nun zwei Möglichkeiten, die negativen Auswirkungen der genannten Einschränkungen zu beseitigen:

- Begrenzung der stochastischen Streuungen auf eine bestimmte Größenordnung durch geeignete elektrotechnische Glieder, wie z. B. Filter.

- Einführen andersartiger Neuronen, die verbesserte Eigenschaften besitzen, d.h. günstigere Runlängencharakteristik und geringere stochastische Streuung der Ausgangsfolgen.

4.3.1 *Online–Training* statt Gegenstrom–Verfahren

Aus Einsparungsgründen bei der Chipfläche wird in der bisher vorgeschlagenen Hardware–Realisierung das sogenannte Gegenstrom–Verfahren verwendet, welches die Stationaritätsbedingung für die Codierung verletzen könnte. Außerdem ist das Training nun nicht mehr echtes *Online–Training*, weil die Gewichte während des Anlegens eines Musters nicht konstant bleiben. Dies könnte beim Training zu einigen unbekannten Nebenwirkungen führen, zumindest ist diese Art von Training in der Literatur bisher nicht bekannt. Um derartige Nebenwirkungen von anderen zu unterscheiden, erscheint es

sinnvoll, für die weitere Analyse statt des Gegenstrom–Verfahrens das traditionelle *Online–Training* zu verwenden. Damit kann einerseits ein Vergleich zur traditionellen Methode durchgeführt und andererseits der Grund für die Divergenz des Trainings bei einigen Beispielen besser festgestellt werden. Verwendung des *Online–Trainings* ist nicht nur bei taktgenauer Software–Simulation, sondern auch bei einer Hardware–Implementierung machbar, sogar unter gewissen Umständen ohne zusätzlichen Platzbedarf auf dem Chip. Nach den Ergebnissen aus dem vorangegangenen Kapitel ist es für die Konvergenz des Trainings nicht entscheidend, ob ein Netz mit oder ohne ADDIEs betrieben wird (siehe Abbildungen 3.7 und 3.8). Aus diesem Grund könnte die Chipfläche des ADDIE–Zählers für einen neuen INDIE–Zähler, nämlich INDIE1, zur Verfügung stehen. INDIE1 ist nur für die Gewichtsmodifikation zuständig. Der bisherige INDIE–Zähler, nun INDIE0, kümmert sich dagegen um alle Gewichtungen. Nach einer gewissen Anzahl von Takten, während der ein Musterpaar an den Netzeingang und -ausgang angelegt wird, wird der Wert des INDIE1–Zählers in den INDIE0–Zähler kopiert. Dann findet das Anlegen eines neuen Muster–Paares statt. Was in diesem Fall an Chipfläche zusätzlich notwendig ist, beschränkt sich auf zusätzliche Leitungen zwischen INDIE0 und INDIE1, welche die Werteübertragung zwischen beiden Zählern übernehmen. Die Abbildung 4.20 zeigt ein Blockschema des modifizierten Synapsenelements. Durch einen Umbau des Synapsenelements läßt sich auch einfach das Batch–Verfahren verwenden, indem die Werteübertragung durch ein Steuerungssignal lediglich nach einer ganzen Epoche erlaubt wird.

Obige Überlegungen werden durch Software–Simulationen bestätigt, indem das Netz unter den gleichen Umständen (d.h. gleiche Aufgabe, gleiche Lernparameter usw.) mit unterschiedlichen Lernstrategien, nämlich Gegenstrom–Verfahren, *Online–Training* und Batch–Verfahren, trainiert wird. Die Abbildung 4.21 zeigt das Ergebnis für das XOR–Problem. In allen Fällen gelangt das Training zur Konvergenz. Das Batch–Verfahren benötigt für das Konvergieren mehr Zeit, weil die Gewichtsmodifikation nur einmal je Epoche stattfindet. Außerdem wird die optimale Abstiegsrichtung wegen der starken stochastischen Streuung meist verfälscht. Aus dieser Sicht ist das *Online–Training* sogar vorteilhafter. Das Verhalten des Trainings ist beim Gegenstrom–Verfahren und beim *Online–Training* fast identisch. Auf Grund der Ergebnisse vieler Software–Simulationen kann als Schlußfolgerung gezogen werden, daß die Verwendung des Gegenstrom–Verfahrens keine negativen Auswirkungen auf die Konvergenz des Verfahrens hat. Deshalb werden alle nachfolgenden Beispiele mit dem Gegenstrom–Verfahren bearbeitet, obwohl das ursprüngliche *Online–Training* sowohl per Software als

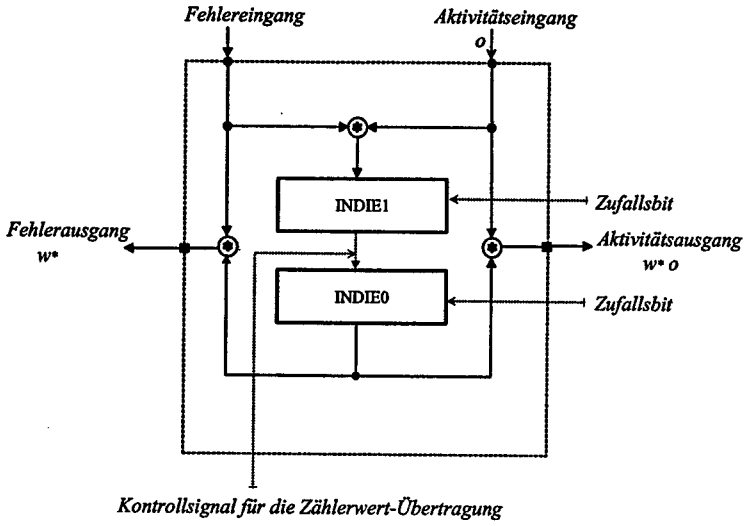


Abb. 4.20: Blockschema des modifizierten Synapsenelements

auch in Hardware mit nur unwesentlich erhöhtem Aufwand implementiert werden könnte.

4.3.2 ADDIE als Tiefpaßfilter zwischen Neuronenschichten

Nach der Analyse im Abschnitt 4.2.2.2 liefert das verwendete Neuron eine Ausgangsfolge mit starkem Rauschen, wenn seine Eingangsfolge einen Wert in der Nähe des Ursprungs repräsentiert. Je steiler seine Aktivierungsfunktion ist, umso stärker ist diese Nebenwirkung, die das Training zur Divergenz bringen kann. Denn das Training wird nun statt durch den Lernalgorithmus, vorwiegend durch Zufall kontrolliert. Im laufenden Abschnitt wird beschrieben, wie diese Nebenwirkung in gewissem Umfang unterdrückt werden kann, damit der Lernalgorithmus beim Training wieder die Hauptrolle übernimmt.

Im Bereich der Elektrotechnik ist das Filtern eine bekannte, ausgereifte und effektive Methode zur Verminderung von Rauschen. Diese Technik kann hier zum Einsatz kommen, wenn ein hierfür geeignetes Filter gefunden werden kann. Außer allgemeinen Eigenschaften sollte das Filter

- leicht in das vorhandene Verfahren integrierbar sein,

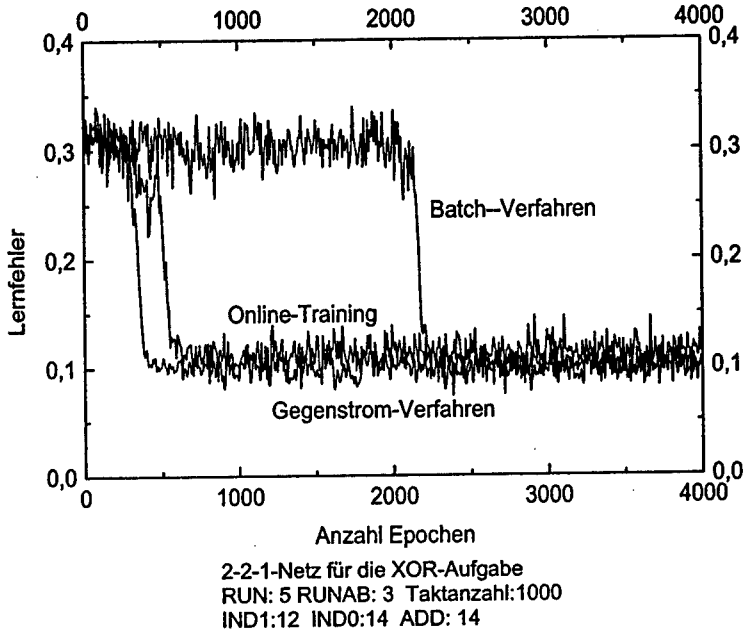


Abb. 4.21: *Konvergenz–Vergleich zwischen Gegenstrom–Verfahren, Online–Training und Batch–Verfahren*

- für seine Implementierung möglichst wenig Chipfläche belegen,
- die Eigenschaft der Ausgangsfolge des S–Automaten nicht weiter verschlechtern.

Aus diesem Grund kommt als Filter das ADDIE in Betracht, weil es ein digitales Tiefpaßfilter (TPF) mit exponentieller Stoßantwort darstellt und so die obigen Bedingungen erfüllen kann. Läßt sich ein ADDIE direkt an den Ausgang eines S–Automaten anschließen, fungiert es wie ein Tiefpaßfilter, welches das Rauschen der Ausgangsfolge des S–Automaten vermindert. Die Abbildung 4.22 zeigt den Verlauf des Ausgangssignals des S–Automaten bei Verwendung der hier erläuterten Filtertechnik.

Zum Vergleich wird das Ergebnis ohne ADDIE als TPF unter sonst gleichen Randbedingungen in der Abbildung 4.23 gezeigt. Es ist deutlich zu sehen, daß die stochastische Streuung, insbesondere in der Nähe des Ursprungs, stark unterdrückt wird. Eine weitere ermutigende Eigenschaft ist, daß sich die stochastische Streuung nicht vergrößert, wenn die entsprechen-

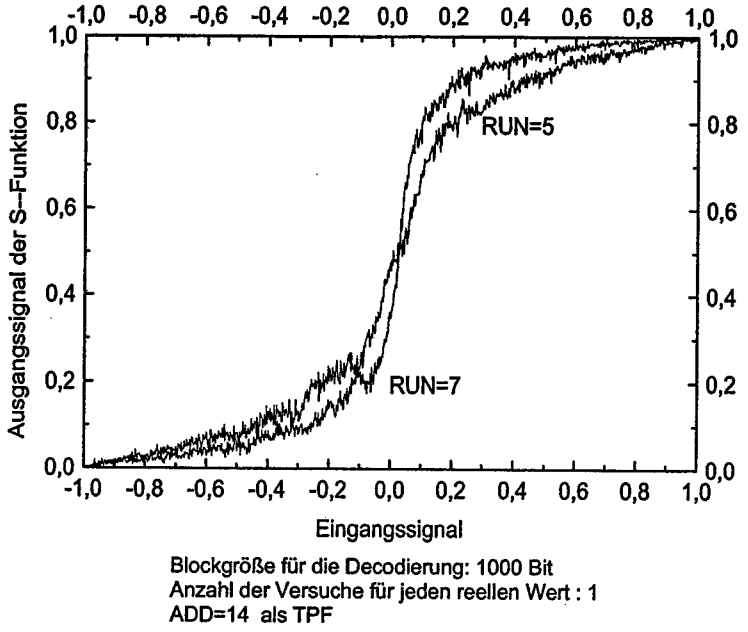


Abb. 4.22: *Verlauf des mit einem ADDIE gefilterten Ausgangssignals des S-Automaten*

de S–Funktion steiler wird, was bei dem Verfahren ohne TPF der Fall ist.

Noch ein anderer Vorteil läßt sich damit gewinnen, nämlich die Verbesserung der Runlängen–Verteilung der Ausgangsfolge. Dank des DSC im ADDIE wird die erwartete Runlängencharakteristik der Bitfolge nach der Codierung wieder hergestellt. Abbildung 4.24 zeigt die Runlängen–Verteilung der Ausgangsfolge, wenn eine 0,5–Folge am Eingang des S–Automaten angelegt wird. Eine solche Runlängen–Verteilung entspricht einer m-Sequenz.

Zur Implementierung dieser Technik wird ein geringer Umbau der vorhandenen Synapse notwendig, der allerdings keine zusätzliche Chipfläche erfordert. Wie in den vorherigen Abschnitten erläutert, hat das als Momentum-Konstante eingesetzte ADDIE wenig Einfluß auf die Konvergenz des Verfahrens. So kann es für den Zweck des TPF weiter verwendet werden, und zwar durch einen Umbau der Verbindungsleitung in der Synapse. Die Abbildung 4.25 zeigt das entsprechende Blockschema.

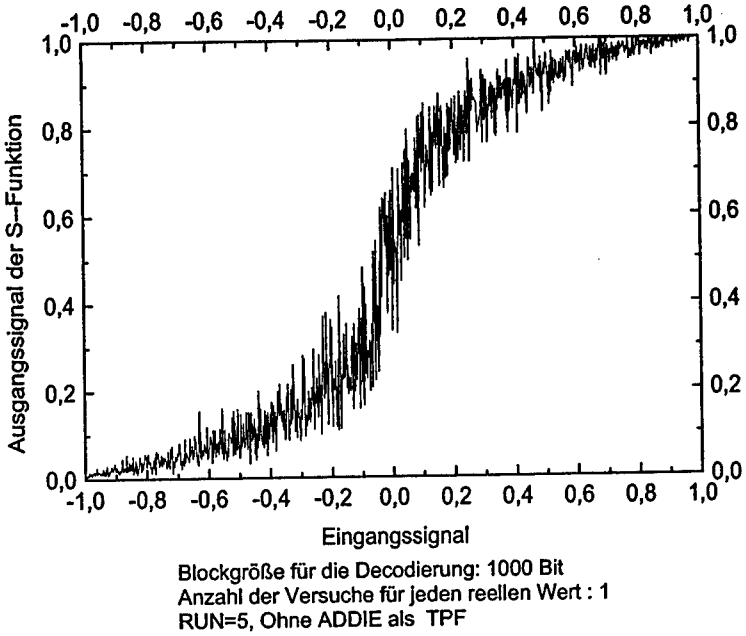


Abb. 4.23: Verlauf des Ausgangssignals des S–Automaten ohne TPF

Die Einsetzbarkeit des ADDIE als TPF ist begrenzt, weil das ADDIE eine Anlaufzeit benötigt, um sich an den neuen Eingangswert anzupassen. In der Schicht, in der das Netz den Zugang zur Außenwelt hat, ändert sich das Eingangssignal beim Wechseln der Trainingsmuster meistens nicht kontinuierlich, wie es z.B. beim Training für das XOR–Problem der Fall ist. In diesem Fall darf das TPF nur in den Synapsen, die zwischen der verborgenen und der Ausgangsschicht liegen, eingesetzt werden[2]. Obwohl die Veränderung des Signals zwischen Neuronenschichten wegen des Musterwechsels am Netzeingang nicht kontinuierlich erfolgt, ist sie jedoch viel kleiner als zwischen Eingangs- und verborgener Schicht. An solche kleinen Änderungen kann sich das ADDIE schnell genug anpassen. Aus dieser Sicht kann die Nebenwirkung der Anlaufzeit des ADDIE vernachlässigt werden. Ein anderer zu beachtender Punkt ist, daß das TPF, sofern ein sequentieller DSC in den Synapsen eingesetzt wird, nichts hilft, weil sich der S–Automat in diesem Fall anders verhält (siehe Abschnitt 4.2.2.1). Wie der sequentielle

[2]Falls von einem zweischichtigen Netz die Rede ist

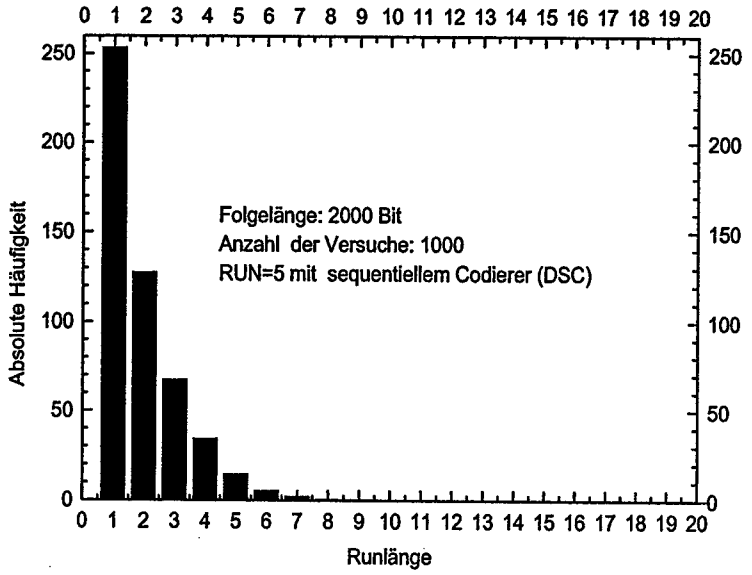


Abb. 4.24: *Runlängen–Verteilung der Ausgangsfolge eines ADDIE als TPF zwischen Neuronenschichten*

DSC trotzdem ohne diese Nebenwirkung angewendet werden kann, soll im folgenden besprochen werden.

Zum Testen wird diese Technik zunächst in dem minimalen MLN gemäß Abbildung 4.15 angewendet, damit ihre Auswirkung durch Kontrolle der Ausbreitung der stochastischen Streuung untersucht werden kann. Aufgrund ihrer Implementierung in der Synapse und wegen der Anlaufzeit kann sie nur zwischen beiden Neuronen eingesetzt werden, und zwar im Gewicht W_1 der zweiten Einheit. Dadurch ist ihre Auswirkung zur Unterdrückung der Ausbreitung der stochastischen Streuung nur im Monitor M_3 (siehe Abbildung 4.15) deutlich zu sehen. Die Abbildung 4.26 zeigt das Ergebnis der Software–Simulation. Man sieht deutlich, daß die Standardabweichung im Fall mit TPF viel kleiner ist als im Fall ohne TPF. Ihre Größenordnung ist fast wie bei einer normalen Codierung. Damit könnte behauptet werden, daß sich die Stärke der stochastischen Streuung im Vergleich zur codierungsbedingten Streuung durch ihre Ausbreitung nicht wesentlich vergrößert.

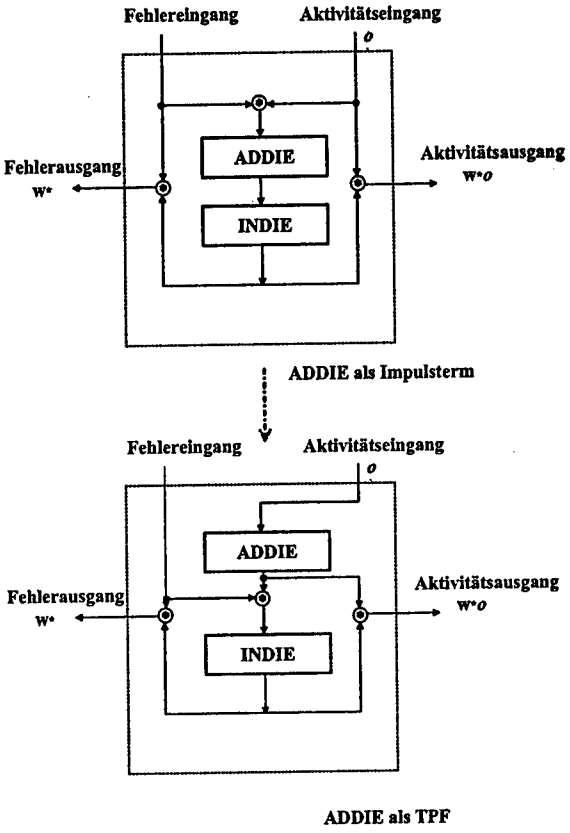


Abb. 4.25: Umbau der Synapse zur Verwendung eines ADDIE als TPF

Nun wird das TPF bei einer kleinen Trainingsaufgabe eingesetzt, um seine Tauglichkeit in der praktischen Welt zu untersuchen. Dafür wird wieder das XOR–Problem herangezogen. Wie immer ist ein 2-2-1-Netz für diese Aufgabe eingerichtet und nur die Synapsen zwischen der verborgenen und der Ausgangsschicht werden für die Anwendung des TPF umgebaut (siehe Abbildung 4.25). Das Ergebnis der Software–Simulation ist in Abbildung 4.27 dargestellt. Der Einsatz des TPF zeigt einigen Zeitgewinn beim Training gegenüber dem Fall ohne TPF. Ein weiterer Vorteil ist, daß bei sehr steiler S–Funktion (großem Wert von RUN) das Training unter Einsatz eines TPF immer noch konvergiert. Ohne TPF war das nicht der Fall. Diese Eigenschaft ist besonders wichtig für große Netze, bei denen steilere S–Funktionen

für die Kompensation des $[\frac{1}{N}]$–Problems erforderlich sind. Ohne TPF liefert eine steile S–Funktion meistens auch sehr starkes Rauschen, besonders in der Nähe des Ursprungs (im M–Bereich).

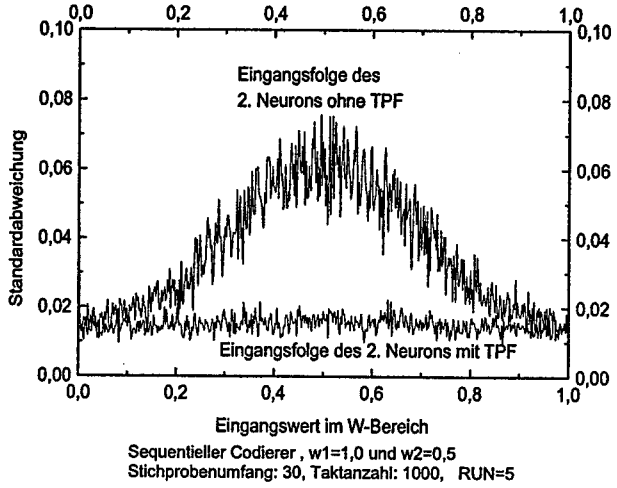


Abb. 4.26: Auswirkung des Einsatzes des TPF in einem MLN

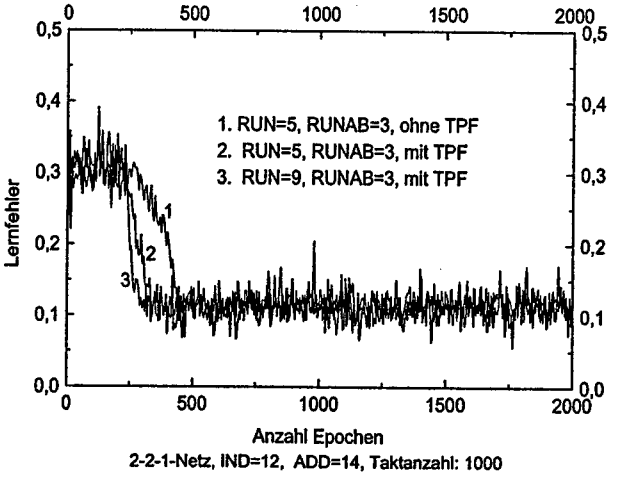


Abb. 4.27: Einsatz des TPF für die XOR–Aufgabe

4.3.3 Einsatz eines modifizierten Neurons

Die gerade erläuterten Maßnahmen im Zusammenhang mit Filterwirkungen nehmen keine zusätzliche Chipfläche in Anspruch, haben aber einige positive Wirkung gezeigt. Jedoch ist die Wirkung entweder relativ gering oder ihre Einsatzmöglichkeit ist begrenzt. Deshalb wird versucht, das vorhandene Neuron durch ein modifiziertes Neuron zu ersetzen, welches ebenfalls eine S–förmige Aktivierungsfunktion liefert, aber bessere Eigenschaften besitzt als zuvor. Außerdem soll es sich leicht in das vorhandene Verfahren integrieren lassen und keine zusätzlichen Parameter für das Training erfordern. Grundlage dieses Neurons ist ein Vorschlag von Rübel [56]. Das zugehörige Blockschema ist in Abbildung 4.28 dargestellt. Wesentliche Bestandteile sind ein Schieberegister, das R_n Bit lang ist, und ein Komparator, der die Anzahl von Einsen im Schieberegister mit einer Schwelle R_t vergleicht. Wird diese Anzahl größer als die Schwelle, wird eine Eins ausgegeben, andernfalls eine Null. Auf dieser Weise wird eine S–förmige Aktivierungsfunktion erzeugt.

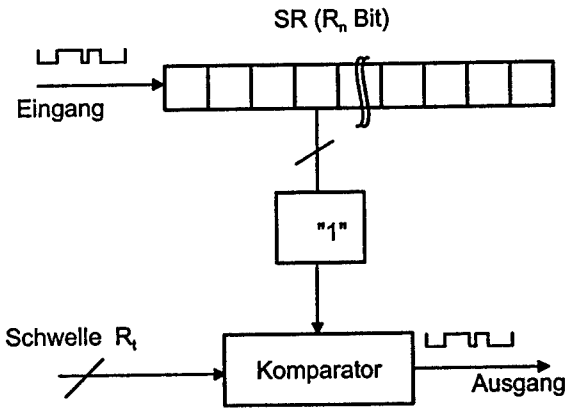


Abb. 4.28: Blockschema nach dem Vorschlag von Rübel [56]

4.3.3.1 Theoretisches Modell und Kennlinie

Sei $\bar{p}$ die Wahrscheinlichkeit, die von einer binären Eingangsfolge B_p repräsentiert wird, τ die Anzahl der Einsen im Schieberegister SR, R_n die Länge des Schieberegisters SR (in Bit) und R_t die Schwelle. τ ist offensichtlich eine Zufallsvariable. Wird B_p eine endlose Bernoulli–Folge, dann besitzt τ eine Binomialverteilung mit den Parametern R_n und $\bar{p}$. Das heißt (siehe

Definition in [15]): $\tau \sim Bi(R_n;\bar{p})$. So läßt sich die Wahrscheinlichkeit P, daß τ gleich einer ganzen Zahl k ist, berechnen:

$$P \equiv P(\tau = k) = \binom{R_n}{k} \bar{p}^k (1-\bar{p})^{R_n-k} \tag{4.17}$$

$$R_n \in \{1,2,3,...\}; k \in \{0,1,...R_n\}; 0 < \bar{p} < 1$$

Dadurch ist die Wahrscheinlichkeit o, daß eine Eins am Ausgang der obigen Schaltung eintritt, ebenfalls leicht zu berechnen:

$$o \equiv P(\tau > R_t) = \sum_{k=R_t+1}^{R_n} P(\tau = k) = \sum_{k=R_t+1}^{R_n} \binom{R_n}{k} \bar{p}^k (1-\bar{p})^{R_n-k} \tag{4.18}$$

$$R_t \in \{0,1,2,...,R_n\}$$

Dies ergibt genau eine S–förmige Funktion von $\bar{p}$, welche den in der Abbildung 4.29 gezeigten Kennlinien zu unterschiedlichen Werten von R_n entspricht. Je größer die Schwelle R_t ist, desto weiter nach rechts schiebt sich die S–Kurve (siehe Teil A in der Abbildung 4.29); je länger das Schieberegister ist, umso steiler ist die erzeugte S–Funktion (siehe Teil B in der Abbildung 4.29). Durch viele Simulationen wurde festgestellt, daß eine sigmoid-ähnliche Funktion erzeugt wird, wenn die Länge R_n des Schieberegisters und die Schwelle R_t folgende Bedingung erfüllen:

$$R_n = 2R_t - 1 \tag{4.19}$$

Kennlinien in Abbildung 4.29 sind auf Basis der Erfüllung dieser empirischen Gleichung erzeugt worden. Dadurch wird nur ein Parameter für die Bestimmung der neuartigen S–Funktion notwendig. In den weiteren Kapiteln dieser Arbeit wird diese empirische Gleichung verwendet. Aus diesem Grund wird die Länge des Schieberegisters immer als ungerade ganze Zahl geführt, so daß die S–Funktion zum Punkt $(0,5)$ symmetrisch wird.

4.3.3.2 Implementierung des modifizierten Neurons

Wie erläutert, sind die wesentlichen Komponenten des modifizierten Neurons das Schieberegister und der Komparator, die natürlich eine bestimmte Chipfläche in Anspruch nehmen werden. Das Schieberegister sollte lang genug sein (32 Bit oder 64 Bit), so daß R_n als Lernparameter in einem sinnvollen Wertebereich eingestellt werden kann. Je nach der Größe von R_n wird das ganze Schieberegister oder ein Teil davon für das Zählen der eintretenden Einsen gebraucht. Der wirklich benötigte Teil wird als **aktueller**

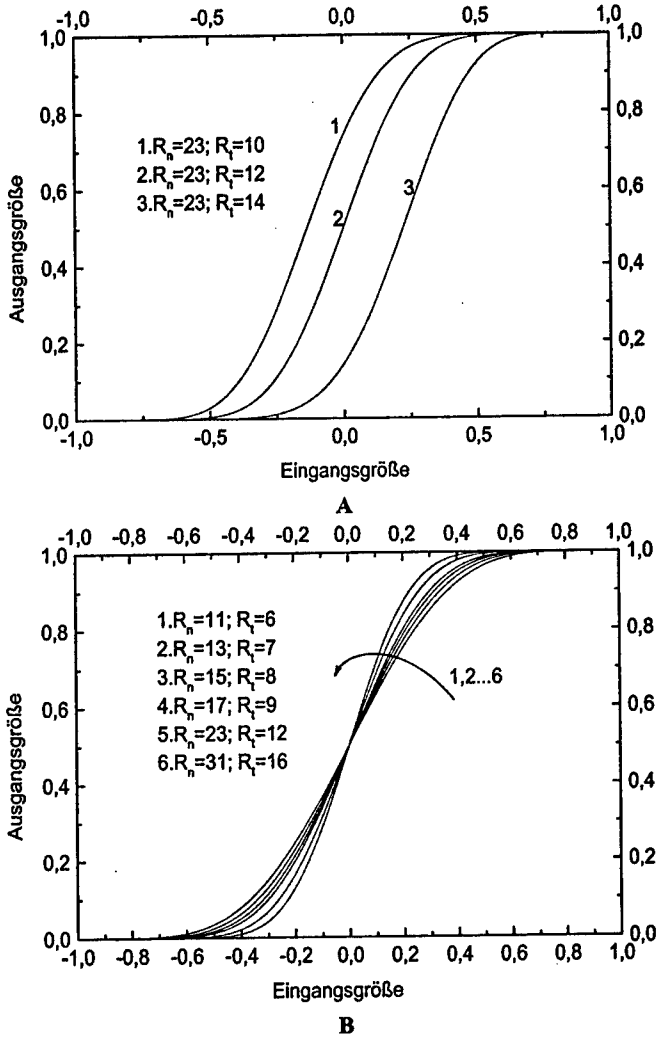


Abb. 4.29: Kennlinien der alternativen S–Funktion mit unterschiedlichen Parametern

Bereich bezeichnet. Für die Implementierung der Komparatoren kann ein Zähler verwendet werden, der ein Bit länger als die Hälfte des Schieberegisters sein soll. Der Zähler wird um Eins inkrementiert oder dekrementiert,

wenn eine Eins in den oder aus dem aktuellen Bereich des Schieberegisters geschoben wird. Wenn der Zähler vorher mit einem gewissen Wert, je nach der Größe der Schwelle R_t, geladen wird, dann kann das Überlauf-Bit des Zählers als Ausgang des Neurons dienen.

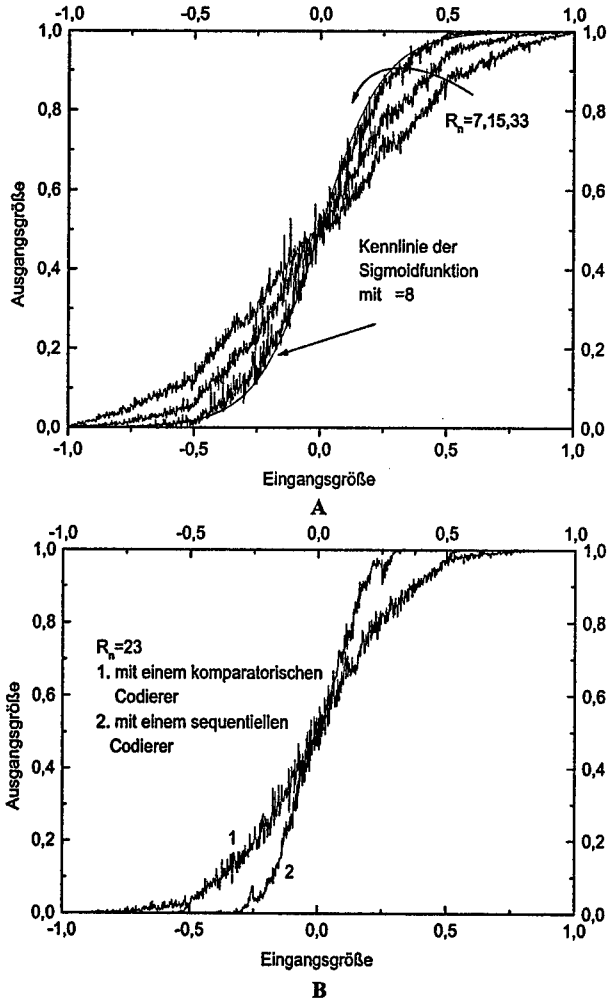


Abb. 4.30: Experimentell ermittelter Verlauf der alternativen S-Funktion für verschiedene Werte des Parameters R_n

Nach diesem Schema wurden Software–Simulationen durchgeführt, deren Ergebnisse in der Abbildung 4.30 gezeigt werden. Die Testbedingung hier ist dieselbe wie in Abschnitt 2.3.3. Das heißt: In der Darstellung wurden 1000 Punkte mit gleichem Abstand im Intervall $[-1, 1]$ abgetastet und zu je einer Bitfolge codiert, und die Ausgaben der Automaten mit unterschiedlichen Werten von R_n wurden dann in Blöcken von 1000 Bit decodiert. Für jeden Punkt auf der Achse des Eingangssignals wird der gleiche Vorgang dabei sechsmal wiederholt, und alle sechs Ausgabewerte der S–Funktion werden dann gemittelt und als ein Punkt in der Kurve aufgetragen.
Verglichen mit der Abbildung 2.8 ist der Verlauf der alternativen S–Funktion deutlich verbessert und fast identisch mit einer konventionellen Sigmoidfunktion der Steilheit μ=8, wenn der Parameter R_n gleich 33 Bit ist (siehe A in der Abbildung 4.30). Unter Verwendung eines sequentiellen Codierers ändert sich der Verlauf nicht sonderlich, wird allerdings steiler. Trotz der großen Steilheit sind die starken Streuungen in der Nähe des Ursprungs nun nahezu verschwunden (siehe B in der Abbildung 4.30). Dies ist besonders günstig für eine Situation, bei der sehr steile S–Funktionen verwendet werden müssen, um die Nebenwirkungen des $\frac{1}{N}$–Mittelungsverfahrens zu kompensieren. Außerdem könnte das Training bei steiler Funktion mit weniger Streuung schneller und mit einem niedrigeren Lernfehler zur Konvergenz kommen.

4.3.3.3 Ableitung der neuen S–Funktion

Im BP–Algorithmus wird die erste Ableitung der nichtlinearen Funktion des Vorwärtsdurchgangs für die Bestimmung der Suchrichtung benötigt. Die Erzeugung einer echten Ableitung der nichtlinearen Funktion ist in der Tat kaum möglich und auch nicht nötig (siehe die Diskussion in Abschnitt 4.2.1.3). Eine Lösung dafür ist, eine Näherung zu benutzen, die zu dem vorhandenen Verfahren paßt und sich leicht implementieren läßt.
Bekannteste und populärste Methode dafür ist die Differenztechnik, bei der sich das Differential durch die entsprechende Differenz ersetzen läßt. Das Schema für die Erzeugung einer Differenz läßt sich mit der Abbildung 4.31 darstellen.
Ähnlich wie bei der Herleitung der S–Funktion läßt sich die theoretische Kennlinie folgendermaßen beschreiben:
Sei δ die Wahrscheinlichkeit, daß die Anzahl der Einsen im aktuellen Bereich des Schieberegisters zwischen $R_t - d$ und $R_t + d$ liegt. Hier ist d eine ganze Zahl, die zwischen 0 und R_t liegen soll. Dann ergibt sich:

$$\delta \equiv P(R_t - d < \tau < R_t + d)$$

$$
\begin{aligned}
&= \sum_{k=R_t-d}^{R_t+d} P(\tau = k) \\
&= \sum_{k=R_t-d}^{R_t+d} \binom{R_n}{k} \bar{p}^k (1-\bar{p})^{R_n-k} \\
&\quad R_n \in \{1,2,3,...\}; k \in \{0,1,...R_n\}; 0 < \bar{p} < 1; \\
&\quad R_t \in \{0,1,2,...,R_n\}; d \in \{0,1,2,...,R_t\}
\end{aligned} \tag{4.20}
$$

Die Kennlinie der nach Gleichung 4.20 berechneten Ableitung der neuen S–Funktion wird in Abbildung 4.32 dargestellt. Es ist deutlich zu sehen, daß die hergeleitete Ableitung einen engen Zusammenhang mit der neuen S–Funktion hat. Je steiler die S–Funktion ist, umso schmaler ist die Ableitung. Der Parameter d ist nun nur für die Einstellung ihrer Höhe zuständig. Dies könnte für die Parameterwahl des Trainings einen Vorteil ergeben, weil sich die Ableitung automatisch an die Änderung der Steilheit der S–Funktion anpassen kann. Bei den alten S– und B–Funktionen ist das nicht der Fall, weil die Form der B–Funktion keinen Zusammenhang mit der S–Funktion hat. Der Parameter RUNAB ist ein selbständiger Parameter, der die Höhe und die Breite der S–Funktion allein bestimmen kann. Die Breite der B–Funktion ist immer eine feste Konstante. Dies kann beim Training zur Verfälschung der Suchrichtung führen, denn der Einflußbereich

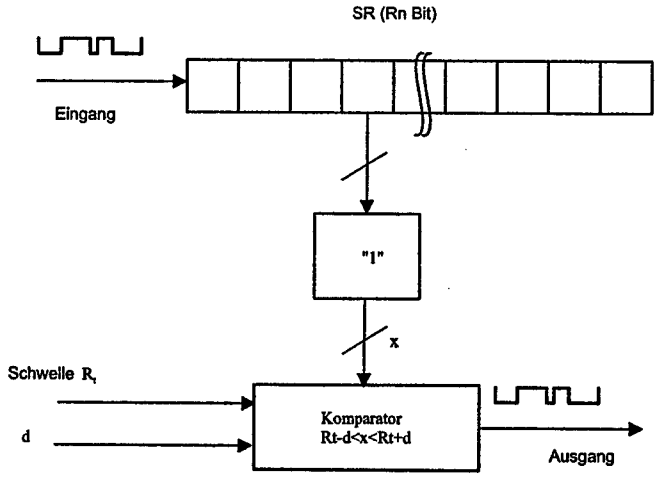


Abb. 4.31: Schema für die Erzeugung der Ableitung der S–Funktion

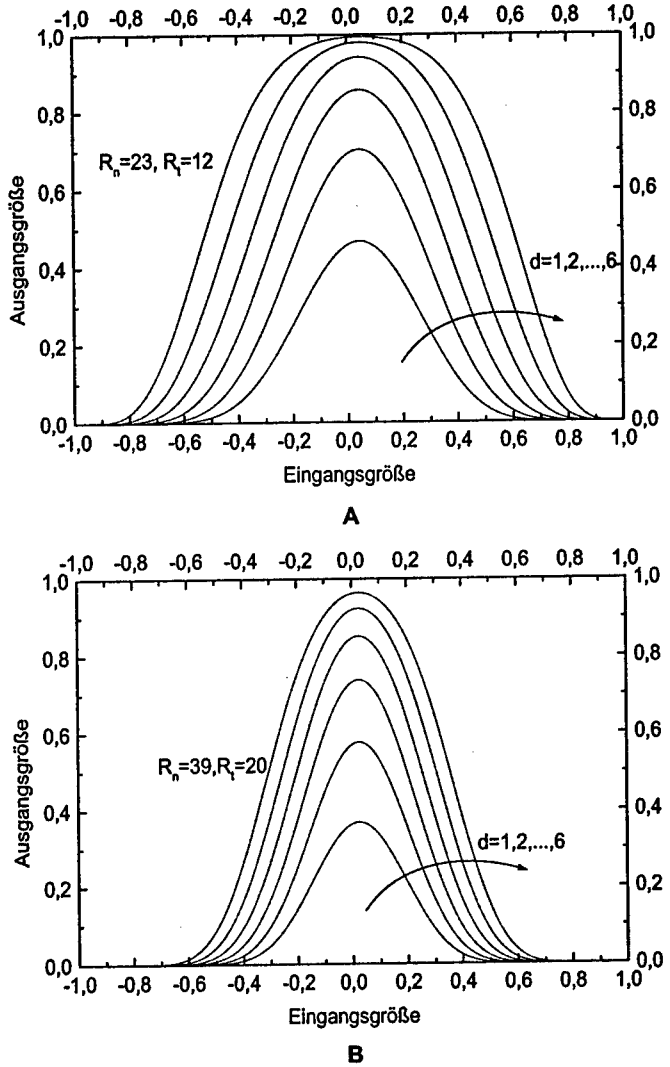


Abb. 4.32: Theoretischer Verlauf der geschätzten Ableitung der alternativen S–Funktion

der S–Funktion ist umso kleiner, je steiler sie ist. Ohne die entsprechende Änderung in der geschätzten Ableitung kann diese Wirkung nicht berücksichtigt werden. Die Einstellung der Höhe der Ableitung allein hilft da

sehr wenig, um darauf zu reagieren. Dafür muß ihre Form, nämlich ihre Breite (oder Schmalheit), einbezogen werden. Aus diesem Grund ist eine gute Kombination von RUN und RUNAB bei den alten Neuronen immer wichtig für die Parameterwahl des Trainings, jedoch schwer in geeigneter Weise festzulegen. Bei den modifizierten Neuronen ist dies relativ leicht, weil sich die geschätzte Ableitung aus der mathematischen Beziehung ergibt und sich so an die Steilheit der S–Funktion anpaßt. Dies kann dazu führen, daß der Trainingsvorgang schneller als zuvor zum Abschluß kommt, weil die jeweilige Suchrichtung besser berechnet werden kann.
Die Implementierung der Ableitung ist fast identisch zu der der S–Funktion. Der tatsächliche Verlauf wird durch Simulationen ermittelt und ist in Abbildung 4.33 veranschaulicht. Im Teil A der Abbildung wurden die Abläufe der Ableitungen für eine neue S–Funktion mit einem relativ kurzen Schieberegister ($R_n = 23$) unter verschiedenen Werten von d gezeigt. Der Teil B zeigt die Verhältnisse bei einer relativ steileren S–Funktion (mit $R_n = 39$).

4.3.3.4 Analyse und Experiment

Die Arbeitsweise des modifizierten Neurons kann man sich so vorstellen, daß eine bitweise Verschiebung eines Fensters auf einer endlosen binären Folge vorgenommen wird. Dabei wird die Anzahl der Einsen im Fenster gezählt und mit einer Schwelle verglichen. Somit handelt es sich nicht mehr um einen Runlängen–Akzeptor wie zuvor. Seine Ausgangsfolge ist erfreulicherweise nicht so stark von der Runlängencharakteristik seiner Eingangsfolge abhängig; sie weicht jedoch von einer m-Sequenz ab. So ist es notwendig, die Runlängencharakteristik der Ausgangsfolge zu studieren. Die Abbildung 4.34 zeigt das entsprechende Ergebnis, wenn dem Neuron eine 0,5–Folge zugeführt wird.
Im Vergleich zu seinem Vorgänger (siehe Abbildung 4.9) ist eine Verbesserung deutlich zu sehen. Dabei gelten die gleichen experimentellen Bedingungen wie zuvor. Die Runlängen–Verteilung entspricht zwar noch nicht der einer m-Sequenz, jedoch befinden sich keine extremen Runs in der Ausgangsfolge. Bei großen Werten von R_n bleibt diese Verteilung unverändert. Dies ist eine gute Eigenschaft für den Einsatz steilerer S–Funktionen in einem großen Netz, die dort unbedingt notwendig sind.
Nun wird ein Blick auf die Ausbreitung der stochastischen Streuungen geworfen. Dabei steht das in der Abbildung 4.15 gezeigte Mini–MLN wieder zur Debatte. Zu betrachten ist die Standardabweichung, wie sie sich vom Eingang bis zum Ausgang des MLN ausbreitet. Das experimentelle Ergebnis wird in der Abbildung 4.35 gezeigt.
Verglichen mit den Kurven in der Abbildung 4.17 ist die Standardabweichung hier erheblich kleiner geworden. Qualitativ beträgt sie nun fast die

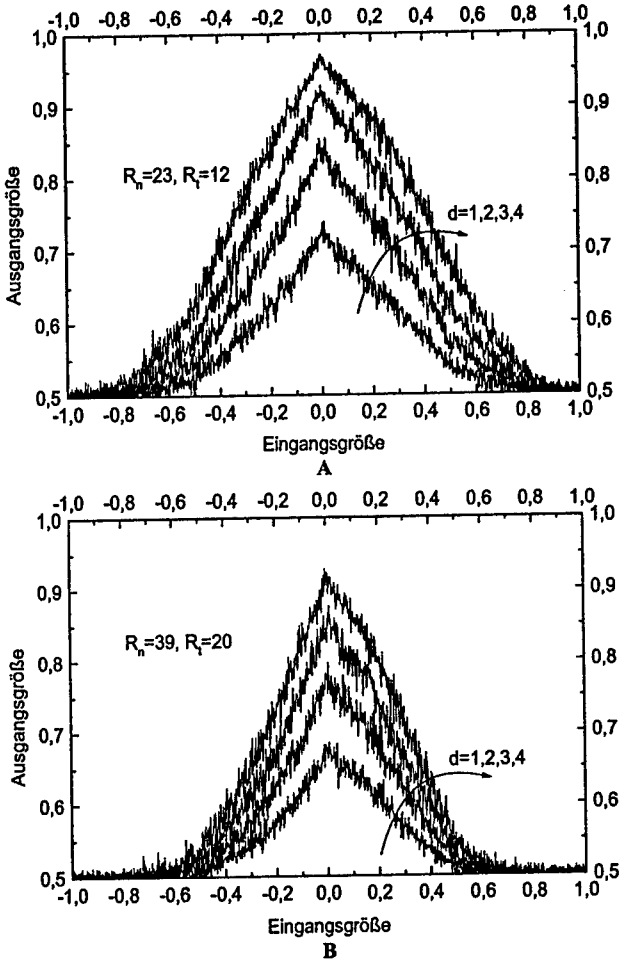


Abb. 4.33: Experimentell ermittelter Verlauf der Ableitung der alternativen S–Funktion

Hälfte wie zuvor, insbesondere in der Nähe des Ursprungs. Wird eine sehr steile S–Funktion angewendet (z.B. $R_n = 39$), vergrößert sich die entsprechende Standardabweichung sehr wenig und liegt immer noch unterhalb von 10%; dies wird meist als Stopbedingung für das Abbrechen des Trainings verwendet.

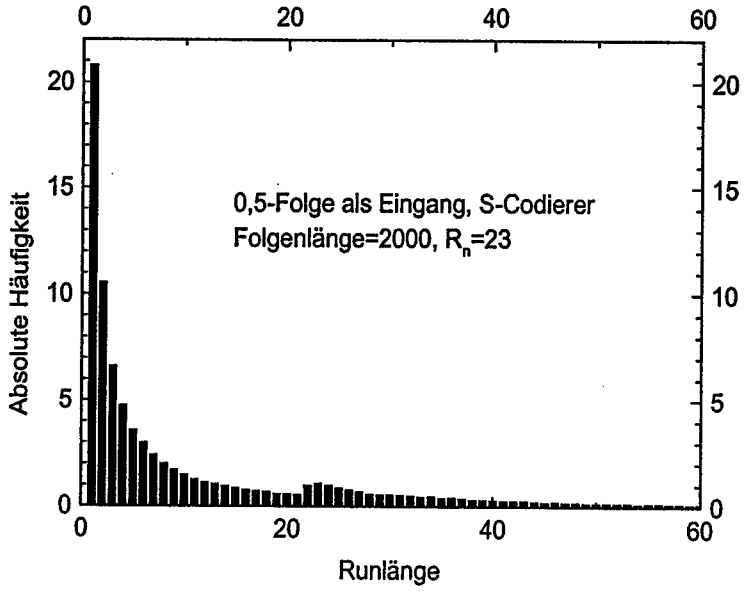


Abb. 4.34: Runlängen–Verteilung des Ausgangsfolge beim modifizierten Neuron

Aufgrund der Anwendbarkeit der steilen S–Funktion erscheint es nun auch möglich, den Trainingsvorgang (durch die Verwendung einer steileren S–Funktion) zu beschleunigen und damit das hier erläuterte Verfahren in vergleichsweise größeren Netzen einzusetzen. Denn eine steile Aktivierungsfunktion kann die Auswirkungen des $\frac{1}{N}$–Mittelungsverfahrens und der $[0,1]$–Einschränkung besser kompensieren. Um diese Vermutung zu bestätigen, wird das modifizierte Neuron zunächst in einer praktischen Aufgabenstellung eingesetzt, nämlich zur Lösung des bekannten XOR–Problems. Weitere Aufgabenstellungen werden in dem folgenden Kapitel anhand mehrerer Beispiele untersucht.

In der Abbildung 4.36 werden die Ergebnisse des Einsatzes der neuen Neuronen im Vergleich zu denen der alten Neuronen dargestellt. Das Resultat deutet an, daß der Trainingsvorgang tatsächlich wie erwartet durch die neuen Neuronen merklich beschleunigt wird. Wegen der unterdrückten Streuungen kann die Suchrichtung bei jeder Iteration besser und genauer bestimmt werden. Dadurch kommt das Training schneller zur Konvergenz.

In der Simulation wird auch festgestellt, daß das neue Neuron gegenüber dem alten noch einen Vorteil hat, nämlich die Unabhängigkeit des Trai-

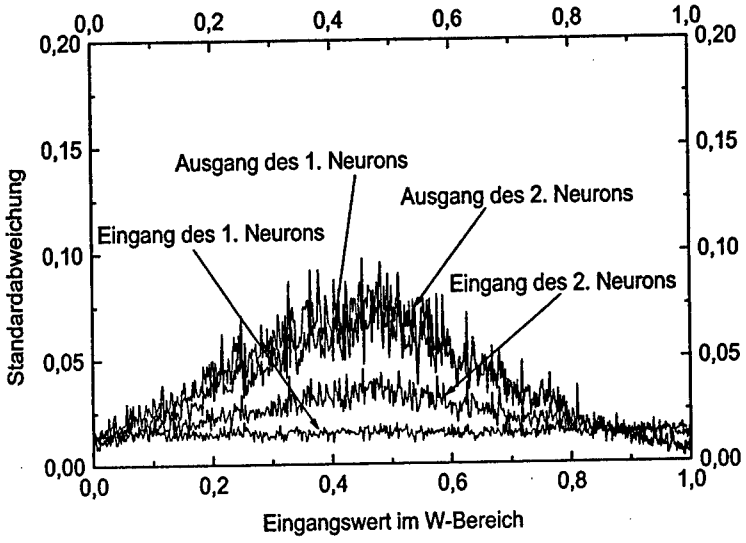


Abb. 4.35: Ausbreitung der Standardabweichung unter dem Einsatz des modifizierten Neurons in einem MLN

nings von der Reihenfolge, nach der die Trainingsmuster ins Netz eingespeist werden. Beim Netz mit den alten Neuronen muß zunächst eine günstige Reihenfolge der Trainingsmuster festgestellt werden; nur dann kann das Training konvergieren. Andernfalls gerät es häufig in die Divergenz. Für das XOR–Problem z.B. konvergiert das Training bei zufälliger Auswahl der Trainingsmuster– einer bekannten Strategie für das Training – meistens nicht. Bei einer relativ komplexen Aufgabenstellung ist die Bestimmung einer geeigneten Reihenfolge der Trainingsmuster genau so kompliziert wie die eigentliche Aufgabenstellung selbst. Insofern ist dies unrealistisch und auch praktisch unmöglich. Mit dem Einsatz des modifizierten Neurons entfällt dieses Problem.

4.3.4 Kurze Diskussion

Die Untersuchungen dieses Abschnitts erlauben folgende Schlußfolgerungen:

- Die Verwendung eines *Online–Trainings* ist zwar technisch machbar und beansprucht kaum zusätzliche Chipfläche, jedoch bringt seine Umsetzung keine wesentliche Besserung im Vergleich zum Gegenstrom–Verfahren beim Training des Netzes. Daher kann festgestellt wer-

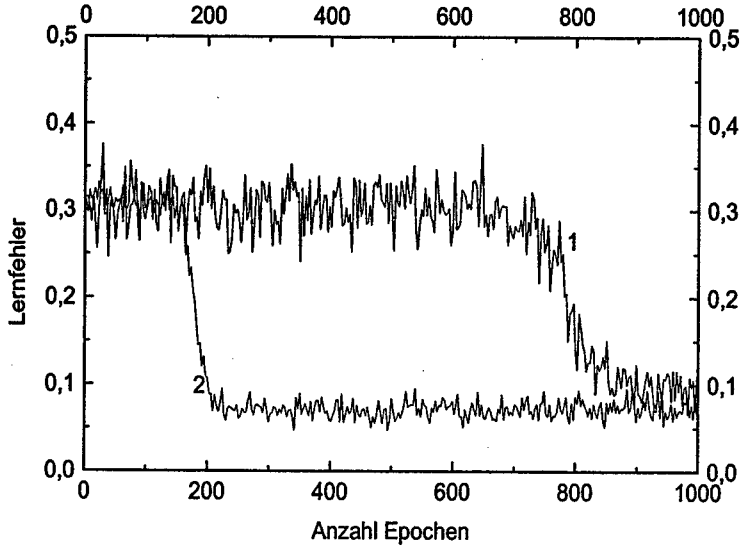


Abb. 4.36: Vergleich der Trainingsergebnisse aus Einsatz der beiden Neuronen für das XOR-Problem

den, daß das Gegenstrom–Verfahren kein besonderes Hindernis für die Skalierung des Verfahrens zu großen Netzen sein sollte. Es ist auch zulässig, es mit den konventionellen Methoden wie beschrieben zu vergleichen.

- Die Verwendung des ADDIE als TPF kann die ungünstige starke Streuung der S–Funktion in der Nähe des Ursprungs im M–Bereich wesentlich unterdrücken. Diese Maßnahme kann leicht ins vorhandene Verfahren integriert werden, und zwar ohne zusätzliche Platzforderungen. Dadurch lassen sich sehr steile S–Funktionen anwenden; dies wird bei ihrem Einsatz in größeren Netzen mit großen N im $\frac{1}{N}$–Mittelungsverfahren erforderlich. Angesichts der Anlaufzeit des ADDIE ist diese Maßnahme allerdings nur für die Zwischenschichten geeignet, bei denen das ADDIE keine direkte Verbindung zur Außenwelt hat.

- Die Arbeitsweise des modifizierten Neurons ist klar und einfach nachvollziehbar. Insbesondere ist sein Verhalten nicht mehr von der Runlängenverteilung seiner Eingangsfolge abhängig. Damit kann der sequentielle Codierer weiterhin für die Codierung der Netzsignale und der Gewichte verwendet werden. Darüber hinaus ist die Eigenschaft seiner Ausgangsfolge deutlich verbessert, nämlich bessere Runlängenverteilung und niedrigere Standardabweichung bei der Ausbreitung der stochastischen Streuung. Dies kann dazu führen, daß das Training schneller oder mit niedrigerem Lernfehler abläuft. Konkretere Aussagen können nur auf der Basis zahlreicher Simulationen in Bezug auf die unterschiedlichen Aufgabenstellungen gemacht werden. Zu erwähnen ist noch ein anderer Punkt, nämlich, daß die genäherte Ableitung einen engen Zusammenhang mit der neuen S–Funktion hat. Durch diesen Zusammenhang kann sie automatisch auf die Änderung der Steilheit der S–Funktion reagieren. Bei den alten Neuronen dagegen steht die B–Funktion in keiner Beziehung zur S–Funktion.

- Stochastische Streuungen bestehen überall in einem Netz mit stochastischen Rechenwerken. Die Standardabweichung kann bis zu 10% der eigentlichen Signalwerte (im W–Bereich) betragen. Aus dieser Sicht könnte das Verfahren für Aufgabenstellungen ungeeignet sein, bei denen der Wertunterschied zwischen den Trainingsmustern an einer Eingangsleitung kleiner als 0,1 (im W–Bereich) ist. Denn das Netz kann bei noch größeren Streuungen solche Unterschiede nicht mehr erkennen. Darüber hinaus kann der Lernfehler durch endloses Training nicht beliebig klein werden, wie es beim konventionellen Verfahren der Fall ist.

4.4 Beurteilung und Testbeispiele

Im folgenden Abschnitt werden die Tauglichkeit und Verwendbarkeit der in den vorherigen Abschnitten erläuterten Techniken zur Verbesserung des Verfahrens anhand verschiedener Beispiele untersucht.

Die Funktionalität des auf stochastischer Rechentechnik beruhenden neuronalen Netzes wird aus Kosten– und Zeitgründen nur in Software implementiert und simuliert. Zu diesem Zweck ist ein Programm notwendig, welches in der Lage ist, die Hardwarevorgänge des Verfahrens so realitätsnah wie möglich zu simulieren (hardwarenahe Software–Simulation). Alle bekannten

und zur Verfügung stehenden NN–Simulatoren (wie SNNS [3], MATLAB von Scientific Computers und Trajan 4.0 Neural Network Simulator von Trajan Software Ltd, Großbritannien) sind für diesen Zweck nicht geeignet, weil sie nur für die konventionellen Fälle konzipiert wurden. Im vorliegenden Fall wurde jedoch ein spezieller NN–Simulator benötigt und entwickelt, um die Ansätze der theoretischen Untersuchungen nachweisen zu können. Der Aufbau und die Bedienung der Simulationssoftware wird im Anhang A in allen Einzelheiten besprochen.

Ein weiteres Ziel der Untersuchungen per Software–Simulation ist, Ansätze darüber zu gewinnen, bis zu welcher Größenordnung ein lernfähiges Netz nach dem hier vorgestellten Verfahren gebaut werden kann und für welche Art von Aufgabenstellungen das Verfahren geeignet ist. Diese Fragen sind nicht leicht zu beantworten, da die Konvergenzeigenschaften eines neuronalen Netzes von der Komplexität der zu lösenden Aufgabenstellung sehr stark beeinflußt werden können.

Normalerweise sind die Dimension der Trainingsmuster und die Anzahl der unabhängigen Trainingsmuster wichtige Faktoren für die Komplexität des Lernvorgangs. Ein neuronales Netz fungiert wie eine Funktion, die einen Eingabevektor auf einen Ausgabevektor abbilden kann. Das Training des Netzes bedeutet, eine solche Funktion zu bestimmen, und zwar repräsentiert durch Funktionsart (Netzarchitektur) und zugehörige Parameter (Gewichte). In diesem Sinne ist das Training eines neuronalen Netzes äquivalent mit der Aufgabenstellung der Interpolation von Meßwerten durch Polynome. Je höher der Grad des Polynoms und je mehr Meßwerte, desto komplexer ist die Aufgabe der Interpolation. Das gleiche gilt auch beim Training eines neuronalen Netzes. Je höher die Anzahl der Neuronen und je mehr unabhängige Trainingsmuster, umso schwieriger ist das Training. So ist die Frage, wie viele Neuronen in einem Netz enthalten sein müssen, um ein bestimmtes Problem zu lösen, identisch mit der nach dem optimalen Grad eines Anpassungspolynoms. Daher kann eine solche Frage nicht unabhängig vom Problem selbst beantwortet werden.

Die Anzahl der Gewichte im Netz wird in den folgenden Abschnitten als ein nützliches Gebrauchsmaß für die Skalierbarkeit des vorgestellten Verfahrens betrachtet. Die Anzahl der Neuronen in der Ausgangsschicht ist von der Aufgabenstellung allein vorgegeben. Daher ist nur die Anzahl der Neuronen in der verborgenen Schicht (wenn vom zweischichtigen Netz die Rede ist) für die Anpassung des Netzes an die Komplexität des Lernproblems zuständig. Aber diese ist unabhängig von der Dimension der Eingabe. Aus diesem

[3] ein bekanntes freies Software–Paket, welches an der Universität Stuttgart für die NN–Simulation entwickelt wurde

Grund ist die gesamte Anzahl der Gewichte im Netz ein besseres Maß als die gesamte Neuronenanzahl, weil sie alle Faktoren, nämlich Dimension der Trainingsmustervektoren und Komplexität des Lernproblems, beinhaltet.

4.4.1 Behandelte Problemklassen und Beispiele

Die Software–Simulation des Verfahrens ist zeitlich sehr aufwendig. So ist die Auswahl der geeigneten Problemklassen keine leichte Aufgabe. In der Literatur ([64], [3],[38], [49] und [43]) und im Internet, (z.B. *UCI machine learning database* (http://www.ics.uci.edu/ mlearn/MLRepository.html) oder *The neural-bench Benchmark collection at Carnegie Mellon University* (http://www.boltz.cs.cmu.edu/) u.s.w.) stehen viele Benchmarks für die Experimente mit ANNs zur Verfügung. Aber die meisten davon sind für den vorliegenden Fall aus den folgenden Gründen nicht geeignet:

- Die Software–Simulation des Verfahrens ist zeitlich sehr aufwendig. Sie muß sich daher auf relativ kleine Aufgabenstellungen beschränken. Viele Benchmarks enthalten hundert oder sogar bis tausend mehrdimensionale Trainingsvektoren. Dies führt zu sehr großen Netzen und astronomischen Trainingszeiten.

- Die meisten Benchmarks beschäftigen sich mit reellen Zahlen, die sich nur um Hundertstel oder sogar Tausendstel voneinander unterscheiden, wie z.B. die Vowels–Daten, deren Aufgabe die Unterscheidung der elf Vokale des Britischen Englisch ist. Wegen der Existenz der stochastischen Streuungen, die größer als der Wert–Unterschied in den Eingangsdaten sein können, stehen die Streuungen während des Trainings im Vordergrund. So kann das Training nie zur Konvergenz kommen.

- Die Benchmarks sind meistens für den Vergleich von Softwarelösungen gedacht, d.h. sie werden für die Bewertung unterschiedlicher Lernalgorithmen verwendet. Das bekannte Zwei–Spiralen–Problem, bei dem zwei ineinander geschachtelte Spiralen in der X–Y–Ebene aus je 97 Trainingspunkten (drei Umdrehungen mit je 32 Punkten pro Drehung plus Endpunkte) in zwei Klassen zu trennen sind, ist ein gutes Beispiel dafür.

Aus diesen Gründen werden nur solche Problemdarstellungen betrachtet, bei denen lediglich binäre Trainingsmuster vorkommen. In der Praxis wurden reell–wertige Zahlen statt reiner ganzer Zahlen (Eins oder Null) für die Komponenten des Trainingsvektors verwendet. Diese Wahl entspricht folgenden Überlegungen:

- Wenn die Trainingsmuster nur Einsen und Nullen enthalten, dann braucht man keinen Codierer für die Eingangs– und Zielmuster. Man

kann dann die Werte innerhalb der $[\bar{N}]$ Takte direkt an die Ein– und Ausgänge [4] anlegen. Die Software–Simulationen in [51] und [16] wurden auf diese Weise durchgeführt. Dabei wurden nur zwei Typen von Bitfolgen am Eingang des Netzes erzeugt: eine mit vollen Einsen (falls die Eingabe eine Eins ist) oder eine mit vollen Nullen (falls die Eingabe eine Null ist). Dieses sind jedoch zwei extreme Fälle für alle stochastischen Rechenwerke im Netz, die bei der stochastischen Technik eigentlich vermieden werden sollten.

- Bei einer zufälligen Gewichtsinitialisierung können manche Gewichte am Anfang des Trainings am Randbereich (Null oder Eins) des Wahrscheinlichkeits–Werteraums liegen. Wenn eine Folge mit vollen Einsen oder Nullen nun ins Netz eingespeist wird, können die Aktivitätswerte einiger Neuronen in der verborgenen Schicht gleich in ihrem Sättigungsbereich landen. Daher nehmen die davor liegenden Gewichte kaum Modifikationen vor. Für das weitere Training wirkt das betroffene Teilnetz fast wie ausgeschaltet. Diese Nebenwirkung kann sich in die nachstehenden Teilnetze ausbreiten, die den Ausgang des scheinbar ausgeschalteten Teilnetzes als ihren Eingang verwenden, wenn die zu ihnen gehörenden Gewichte während des Trainings im Randbereich (Null oder Eins) des Werteraums der Wahrscheinlichkeit landen.

- Viele Aufgabenstellungen in der Wirklichkeit führen nicht zu rein binären Vektorkomponenten. Daher liegen die Komponenten der Trainingsvektoren nach ihrer Normierung meistens im Intervall [0,1] des reellen Zahlenraums. Um die Tauglichkeit des Verfahrens zu testen, ist es daher sinnvoll, die in dem Wahrscheinlichkeitsraum liegenden reellen Werte zu verwenden statt die binären Werte Eins und Null.

- Mit reellen Werten kann man selbst leicht eine Testmenge von Vektoren herstellen, die von den vorgegebenen Vektoren etwas abweichen. Dies ist besonders sinnvoll, wenn die Anzahl der zur Verfügung stehenden Proben sehr gering ist (z.B. das XOR–Problem hat nur vier Proben im binären Fall zur Verfügung, die sowohl als Trainingsmuster als auch als Testmuster verwendet werden müssen).

Die Ausgaben des Netzes sind nach ihrer Decodierung für alle Aufgabenstellungen immer reelle Werte, und zwar liegen sie im Intervall [0,1] (betrachtet als Wahrscheinlichkeit) oder [-1,1] (betrachtet als Maschinenvariable). Diese

[4] Das Anlegen des Zielmusters am Ausgang dient zum Vergleichen der Soll– und Ist–Ausgaben

Netzausgabe–Werte werden bei der Testphase einer binären Aufgabenstellung nach der üblichen 40–20–40–Regel bewertet, d.h. wenn der Ausgabewert des Netzes im Intervall [0,0.4] oder [0.6,1] liegt, wird er als Null oder Eins bewertet. Liegt er im anderen Intervall, nämlich [0.4,0.6], wird er als ungewiß betrachtet. Auf dieser Weise kann leicht festgestellt werden, ob das trainierte Netz den Test besteht oder nicht.

4.4.1.1 Das Parity–Problem

Beim Parity–Problem soll ein ANN lernen, bei einer ungeraden/geraden Anzahl der Einsen an seinem Eingang eine Eins/Null an seinem Ausgang zu liefern. Das bekannte XOR–Problem ist eigentlich ein Sonderfall davon, bei dem die Eingangsmuster lediglich zweidimensional sind. Aus dieser Sicht ist es auf keinen Fall eine leichte Aufgabe für das Verfahren. Die Datensätze wurden den *UCI Machine Learning Databases* entnommen.
Je größer die Dimension der Eingangsmuster wird, desto größer wird die Anzahl der Gewichte des zu trainierenden Netzes und somit auch die Anzahl der zu klassifizierenden Trainingsmuster. Bei wachsender Dimension des Parity–Problems steigt der Schwierigkeitsgrad der Lernaufgabe enorm. Z.B. bei der Dimension 2 muß das Netz nur vier gerade Linien als Trennlinien im Vektorenraum der Dimension 2 feststellen, wobei schon bei der Dimension 4 das Netz 16 Hyperebenen in einem Vektorenraum der Dimension 4 herausfinden muß.
Der Grund, dieses Lernproblem hier als Beispiel zu nehmen, besteht darin, daß die Fähigkeit des Verfahrens beim Umgang mit relativ schwierigen Lernproblemen nachzuweisen ist. Als Kürzel wird für das entsprechende Lernproblem P_d verwendet. Der Index d steht dabei für die Dimension der Eingabevektoren. Daher steht P_n für das n–Bit–Parity–Problem, wobei $n = 2$ dem bekannten XOR–Problem entspricht.

4.4.1.2 Buchstabenerkennung über Bitmap–Repräsentation

Dieses Beispiel wurde aus der Literatur [17] übernommen. Hier handelt es sich um die Erkennung der Buchstaben, die durch eine Bitmap (z mal s Pixel, z und s =1,2,...) repräsentiert werden. Das Training dient dazu, dem Netz beizubringen, die entsprechenden ASCII–Codes (7 Bit) in Binärdarstellung auszugeben, wenn ihm die Bitmap–Darstellung eines Buchstabens präsentiert wird. Derartige Aufgabenstellungen beziehen sich auf sogenannte OCR–Anwendungen (*Optical Character Recognition*).
Mit diesem Beispiel kann man die Skalierbarkeit des Verfahrens qualitativ untersuchen. Man kann die Anzahl der zu erkennenden Buchstaben festlegen und die Größe der Bitmaps Schritt für Schritt vergrößern, wie z.B. 7*5–Matrix, 9*7–Matrix, 10*8–Matrix, usw. bis zu der Bitmapgröße, bei der das

Training nicht mehr konvergiert. Auf diese Weise können die beiden wichtigen Faktoren der Komplexität des Lernproblems, nämlich die Dimension des Trainingsmusters und die Anzahl der unabhängigen Trainingsmuster, getrennt berücksichtigt werden. Dadurch kann man einen Ansatz gewinnen, ob die Divergenz des Trainings von dem $\frac{1}{N}$–Mittelungsverfahren oder der Komplexität des zu lösenden Problems selbst verursacht wurde. Erhöht sich die Anzahl der unabhängigen Trainingsmuster, nimmt der Schwierigkeitsgrad ebenfalls zu, weil das Netz mehr Klassen als zuvor klassifizieren muß. Das heißt, mehr Klassengrenzen sind zu ermitteln. Um dies zu erreichen, müssen der verborgenen Schicht mehr Neuronen hinzufügt werden. Dadurch nimmt auch die Anzahl der Gewichte zu. In diesem Fall kann man nicht entscheiden, ob das Verfahren selbst, z.B. das $\frac{1}{N}$–Mittelungsverfahren oder die Komplexität des Problems, die Ursache der Divergenz ist. Aus diesem Grund ist es sinnvoll, mit einer festen Anzahl zu trennender Klassen anzufangen. Zunächst wird eine Untermenge des Alphabets für das Klassifizieren gewählt, nämlich A,C,E,Q,S,T,Y. Deren Bitmap–Repräsentationen soll das Netz durch erfolgreiches Training in den entsprechenden ASCII-Code umsetzen.
Im folgenden Abschnitt wird dieses Beispiel als BE_{z*s} bezeichnet. $z*s$ bezeichnet die Größe der Bitmap–Repräsentation der zu erkennenden Buchstaben, z.B. BE_{7*5}, BE_{9*7}, BE_{10*8}, usw. Die Klassenanzahl wird berücksichtigt, indem eine in den Klammern stehende ganze Zahl dahintergesetzt wird. Zum Beispiel steht $BE_{7*5}(7)$ für die Buchstabenerkennung mit sieben zu trennenden Klassen und eine $7*5$–Matrix als Eingangsvektoren.

4.4.1.3 Buchstabenerkennung via numerische Merkmale

Im Vergleich zum Bitmap–Ansatz läßt sich die Buchstabenerkennung auch auf andere Weise behandeln. Einem ANN werden nicht direkt binäre Bitmaps der zu erkennenden Buchstaben präsentiert, sondern Merkmalsvektoren, die durch Vorverarbeitung aus originalen Bitmaps erzeugt werden. Die Merkmale sind unabhängig von der Größe der Bitmaps und der unterschiedlichen Fonts.
In [61] wurden 16 Merkmale zusammengestellt, welche die Erkennungsaufgabe gut erfüllen können. Sie sind numerisch und bestehen aus statistischen Momenten und der Kantenanzahl (*edge counts*) der Bitmap–Darstellung. Das Zielmuster sind 26 englische Buchstaben (von A bis Z), die aus 20 unterschiedlichen Fonts stammen. Durch stochastische Verzerrung wurde ein Datensatz von 20.000 Fällen erzeugt und zu einer Menge natürlicher Zahlen {0,1,...,15} umskaliert. Die fertigen Datensätze kann man vom Internet (ftp://ftp.ics.uci.edu/pub/machine-learning-databases/letter-recognition/) herunterladen.

Um festzustellen, wie das hier erläuterte Verfahren mit nichtbinären Datensätzen umgeht, ist dieser Datensatz geeignet. Erstens ist die Anzahl der unabhängigen Trainingsmuster (sechsundzwanzig) und die Dimension der Trainingsmuster (sechzehn für den Eingang und Eins für den Ausgang) festgelegt. Dadurch ist die Netzgröße auch fast wie vorgeschrieben. Zweitens bleibt das zu trainierende Netz relativ klein. So ist der Trainingsaufwand nicht sehr groß und man kann in einem vernünftigen Zeitraum mehrfach Trainingsversuche durchführen.
Es wurden zur Anpassung an das Verfahren einige Änderungen vorgenommen, nämlich die Konvertierung der Datensätze in die binäre Form. Dazu wurde jede Komponente des Eingangsmusters durch fünfzehn geteilt (Normierung) und jeder der Zielbuchstaben (charakteristisches Format) in eine fünfstellige binäre Zahl (weil $2^5 = 32$) umgewandelt. Damit kann das Simulationsprogramm die Datensätze dieses Beispiels leicht verarbeiten. Als Vereinbarung für die folgende Erläuterung wird das betreffende Beispiel mit BEN_s (s bezieht sich auf die Anzahl der zu erkennenden Buchstaben) bezeichnet.

4.4.2 Ergebnisse und Diskussion

In diesem Abschnitt werden die Ergebnisse der Software–Simulation erläutert, die mit den obigen Beispieldatensätzen durchgeführt wurden. Anschließend wird diskutiert, ob die im letzten Kapitel erläuterten Ansätze verallgemeinert werden können. Zum Schluß wird eine Schätzung oder eine Vermutung über die Skalierbarkeit und Anwendbarkeit des Verfahrens geäußert.

4.4.2.1 Gegenstrom–Verfahren

Es wurde in Abschnitt 4.3.1 durch das XOR–Problem bestätigt, daß die Verwendung des Gegenstrom–Verfahrens anstatt des *Online–Trainings* die Konvergenzeigenschaften des Trainings kaum beeinflußt. Durch Simulationen mit den gerade vorgestellten Beispielen wurde diese Behauptung weiter bestätigt (siehe Abbildungen 4.37 und 4.38).
Bei allen Fällen führt das *Online–Training* etwas schneller zur Konvergenz als das Gegenstrom–Verfahren. Der Unterschied ist nicht sehr groß und kann daher vernachlässigt werden. Wenn das Training durch Hardware (neuronale Chips) durchgeführt wird, spielt ein solcher Unterschied keine Rolle mehr. Die Ergebnisse aller hier erläuterten Beispiele weisen auf, daß die Verwendung des Gegenstrom–Verfahrens im Allgemeinen keine negativen Auswirkungen auf die Konvergenzeigenschaften der Lernalgorithmen ausüben kann.

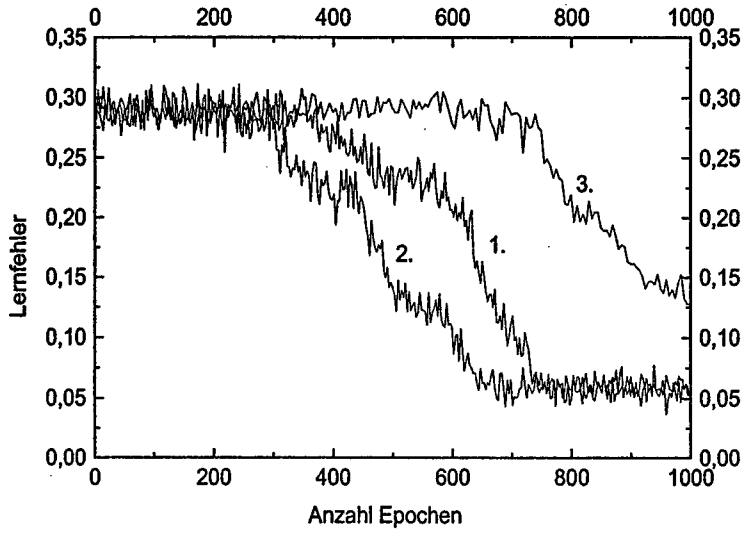


Abb. 4.37: Vergleich der verschiedenen Trainingsverfahren mit Beispiel P_3

4.4.2.2 Verbesserte Konvergenzeigenschaften durch modifizierte Neuronen

Im vorherigen Abschnitt wurde vermutet, daß der Einsatz der in dem Abschnitt beschriebenen modifizierten Neuronen eine Verbesserung der Konvergenzeigenschaften des Verfahrens erbringen könnte. Diese Vermutung wird hier durch obige Beispiele bestätigt. Die Ergebnisse der durchgeführten Simulationen zeigen deutlich einige Verbesserungen im Vergleich zu den alten Neuronen (siehe Tabelle 4.1).

Während das Training unter Verwendung von Netzen mit den ursprünglichen, „alten" Neuronen bei den meisten Beispielen zur Divergenz führt, bewirkt der Austausch gegen die verbesserten, „neuen" Neuronen überwiegend Konvergenz. Die Einzelergebnisse sind in Tabelle 4.1 ausführlich dargestellt.

Das Zahlenpaar s/v in den Spalten „alte/neue Neuronen" deutet an, daß innerhalb von v Versuchen das Training s–mal mit Erfolg (Konvergenz) abgeschlossen wurde. Aus Gründen des großen Zeitaufwands der Simulation wurden bei jedem Versuch nur 1000 Lernzyklen durchgeführt. Die Anzahl der Versuche ist für eine statistisch gesicherte Aussage noch zu gering. Jedoch ist der Trend zur Verbesserung bereits deutlich zu erkennen.

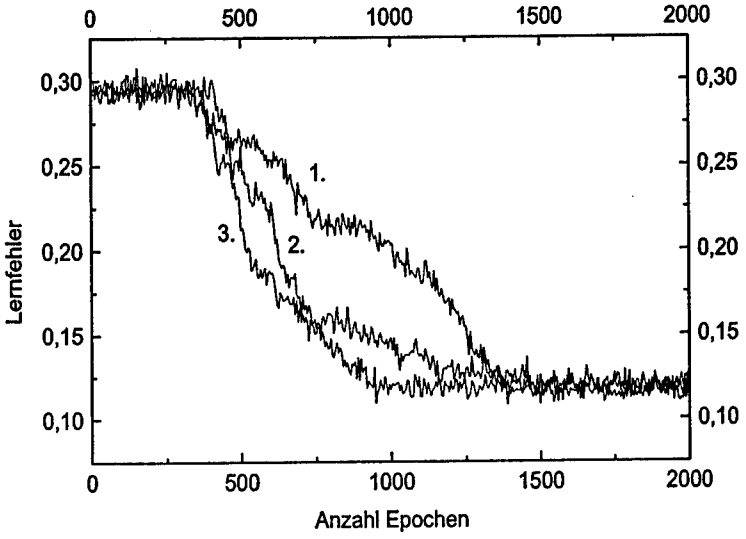


Abb. 4.38: Vergleich der verschiedenen Trainingsverfahren mit Beispiel P_4

Beispiel	*G.Z./N.T.*	*KZ*	*Alte Neuronen*	*Neue Neuronen*
P_3	21/3-4-1	2	2/10	10/10
P_4	121/4-20-1	2	0/10	8/10
$BE_{7*5}(7)$	867/35-20-7	7	0/5	5/5
BEN_4	709/16-32-5	4	0/5	1/5

G.Z. ist eine Abkürzung für Gewichtsanzahl, N.T. für Netztopologie und KZ für Klassenanzahl

Tabelle 4.1: Vergleich der Trainingserfolge mit alten/neuen Neuronen

Bei den alten Neuronen spielt die Reihenfolge der präsentierten Trainingsmuster eine Rolle; sie hat Einfluß darauf, ob das Training zur Konvergenz/Divergenz führt oder nicht. Sogar bei dem XOR–Problem, wo es nur vier Trainingsmuster gibt, kann eine falsche Reihenfolge zum erfolglosen Training führen. Diese Eigenschaft hat mindestens folgende Nachteile:

- Verzicht auf zufällige Auswahl des Trainingsmusters aus der vorgegebenen Trainingsmenge, obwohl dies die übliche Methode in der Lite-

ratur und eine Maßnahme gegen das Steckenbleiben des Trainings in der Nähe eines lokalen Minimums ist.

- Die Festlegung einer richtigen Reihenfolge ist in der Praxis kaum möglich, wenn die Anzahl der zur Verfügung stehenden Trainingsmuster groß wird.

Dagegen ist das Training mit dem Einsatz von neuen Neuronen völlig unabhängig von der Reihenfolge der eingespeisten Trainingsmuster. Diese Eigenschaft hat sich bei allen obigen Beispielen aufzeigen lassen. Dadurch ist man in der Lage, die übliche Methode zu verwenden, nämlich eine zufällige Reihenfolge für die Präsentation der Trainingsmuster. In diesem Fall sind die obigen Nachteile natürlich verschwunden.

Bei der Auswahl der Trainingsparameter wurden die Ansätze aus den vorherigen Kapiteln für alle obigen Beispiele angewendet. Dies hat bei der Suche nach richtigen Parameterkombinationen viel Zeit erspart. Aus den Erfahrungen mit dem alten Neuron ist die richtige Wahl des Wertes für das INDIE das wichtigste (siehe Abbildung 3.7) für die Konvergenz des Trainings überhaupt. Mit anderen Worten reagiert der Lernvorgang sehr empfindlich auf die Zählerlänge des INDIE.

Mit dem Einsatz der modifizierten Neuronen ist dies nicht mehr der Fall. Der wählbare Wertebereich ist größer als zuvor, d.h. das Verfahren reagiert auf diesen Lernparameter nicht mehr so empfindlich wie zuvor. Dies ist besonders günstig für eine Hardware–Implementierung des Verfahrens, weil die Hardwarebestandteile der Parameter wie INDIE– bzw. ADDIE–Länge nicht so leicht oder nicht so häufig geändert werden können wie in der Software.

Die Abbildung 4.39 zeigt den Vergleich der für die Konvergenz wählbaren Zählerlänge des INDIE bei Einsatz der alten und neuen Neuronen. Im Fall der neuen Neuronen liegen die für die Konvergenz wählbaren Werte fast genau innerhalb des Bereichs zwischen ihren theoretischen Unter– und Obergrenzen, die im Abschnitt 3.3.2 mathematisch abgeleitet wurden.

Die gleiche Simulation wurde mit den Beispielen P_3 und P_4 durchgeführt. Die Abbildung 4.40 zeigt die Ergebnisse der Simulation. Wegen des großen Zeitaufwands besteht jeder Versuch hier nur aus 1000 Zyklen. Die einstellbare Zählerlänge des INDIE, mit denen das Training innerhalb von 1000 Zyklen konvergiert (d.h. die Stopbedingung ist erfüllt), ist sehr ähnlich (d.h. mit INDIE=11,12,13 kann das Training innerhalb von 1000 Zyklen konvergieren), obwohl die zur Konvergenz benötigten Lernschritte (Zyklen) unterschiedlich sind. Dies läßt sich auch nachvollziehen: je schwieriger die Aufgabe ist, desto mehr Lernschritte benötigt das Training bis zur Konvergenz.

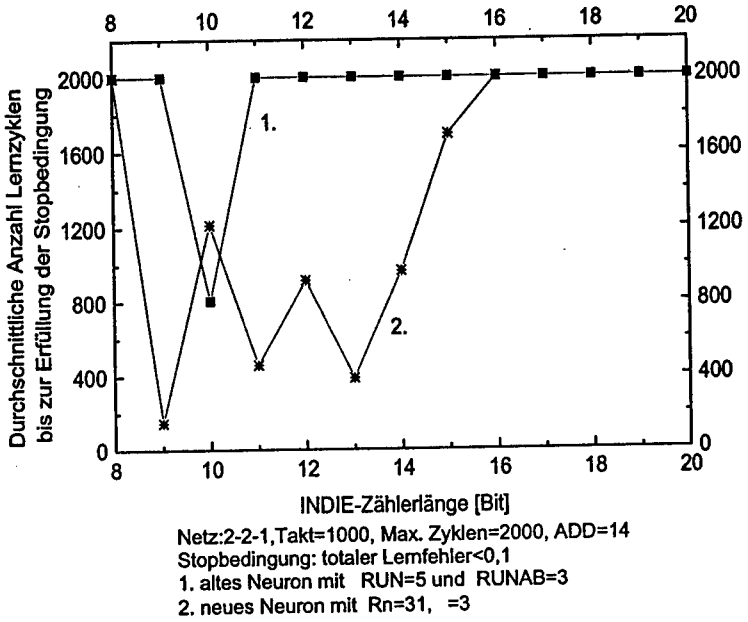


Abb. 4.39: Vergleich der Empfindlichkeit des Verfahrens bzgl. der Zählerlänge des INDIE für Beispiel P_2

4.4.2.3 Skalierbarkeit

Mit dem Beispiel BE_{z*s} läßt sich die Skalierbarkeit des Verfahrens qualitativ gut analysieren, indem die Anzahl der zu erkennenden Buchstaben festgelegt wird und man die Größe der Bitmaps Schritt für Schritt vergrößert. Auf diese Weise nimmt die Anzahl der Gewichte in dem dafür aufgebauten Netz zu, während der Schwierigkeitsgrad des zu lösenden Problems fast unverändert bleibt (weil zumindest die Anzahl der zu suchenden Trennlinien im Vektorraum der Trainingsmuster dadurch nicht anwächst). Man kann den Vorgang umgekehrt geschehen lassen, um die sogenannte Netzkapazität (siehe [55]) eines fertig gestalteten Netzes abzuschätzen, d.h. mit einer festen Anzahl der Neuronen im Netz läßt sich die Klassenanzahl (die Anzahl der zu erkennenden Buchstaben) vergrößern, bis dann das Netz nicht mehr konvergiert.

Zu Anfang wurde ein 35-20-7-Netz für das Beispiel $BE_{7*5}(7)$ gebaut. Damit bestehen 867 Gewichte im Netz. Danach wurde das Netz zu einem 63-32-7–Netz für das Beispiel $BE_{9*7}(7)$ ausgebaut, was 2279 Gewichte im

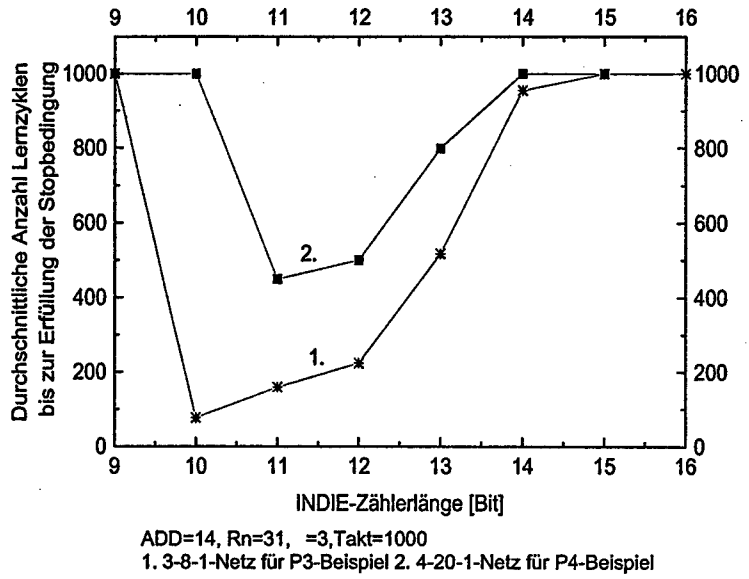


Abb. 4.40: Empfindlichkeit des Trainings bzgl. der Zählerlänge des INDIE als Lernparameter bei den Beispielen P_3 und P_4

Netz ergibt. Für das Beispiel $BE_{10*8}(7)$ ist ein 80-32-7–Netz zum Einsatz gekommen, welches insgesamt 2823 Gewichte besitzt. Die Abbildung 4.41 zeigt die Ergebnisse der Software–Simulationen.

Aus der Abbildung 4.41 ist zu erkennen, daß die Zunahme der Netzgröße keinerlei negative Auswirkungen auf der Konvergenz des Trainings ausübt, solange sich der Schwierigkeitsgrad des zu lösenden Problems nicht ändert.

4.4.2.4 Nichtbinäre Aufgabenstellungen

Alle obigen Erfolge wurden mit binären Beispielen erreicht. Dagegen hat das Training mit nichtbinären Aufgaben, wie Beispiel BEN_s, keine Erfolge gezeigt. Sogar mit $s = 4$ (mit nur vier Buchstaben T,I,D,N und einem 16–32-5–Netz) führt das Training zur Divergenz, obwohl die gleiche Aufgabe durch den traditionellen BP–Algorithmus leicht erledigt werden kann. Die Abbildung 4.42 zeigt das Trainingsergebnis eines traditionellen BP–Netzes für das Beispiel BEN_{52}. Es ist deutlich zu sehen, daß die Konvergenz des Trainings sehr schnell erreicht wird.

Von daher läßt sich vermuten, daß die hier erläuterte Technik für die nichtbinären Anwendungen nicht geeignet ist. In binären Fällen sind die zu

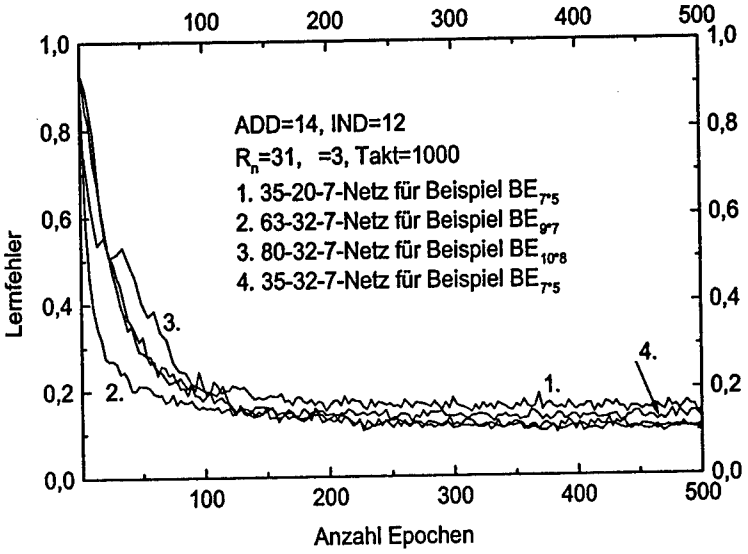


Abb. 4.41: Trainingsergebnisse für Beispiele BE_{z*s} mit unterschiedlichen Größen

bearbeitenden Werte wegen der stochastischen Codierung zwar nicht nur Einsen oder Nullen, aber sie liegen in der Nähe von Eins oder Null. Mit anderen Worten ist eine Abweichung von Eins oder Null der normale Fall. Gibt es ein Maß dafür, wie groß diese Abweichung sein könnte? Um die Diskussionen weiter führen zu können, werden die folgenden Begriffe benötigt. Dabei werden alle Werte nur im Wahrscheinlichkeitsbereich betrachtet, wenn es nicht explizit anders ausgesprochen wird.

Sei $\vec{x} = (x_1, x_2, ..., x_n)$ ein Trainingsvektor

$D(u,v)$ **Absoluter Abstand** zwischen den reellen Werten u und v, d.h. $D(u,v) = |u - v|$

$\theta_{\vec{x}}(0,1)$ **Maximale Abweichung** des Vektors $\vec{x}$ vom Wertebereichsrand.

$$\theta_{\vec{x}}(0,1) = 2 \max_i (\min_{j=0,1} D(x_i, j)) \tag{4.21}$$

Angesichts der obigen Definitionen kann $\theta_{\vec{x}}(0,1)$ nur im Intervall [0,1] liegen. Wenn alle Komponenten des Vektors $\vec{x} = (x_1, x_2, ..., x_n)$ binäre Werte sind, hat $\theta_{\vec{x}}(0,1)$ den Wert Null. Dagegen wird $\theta_{\vec{x}}(0,1)$ Eins, wenn eine der Komponenten den Wert 0,5 annimmt. Dadurch hat man ein Maß bekommen, mit

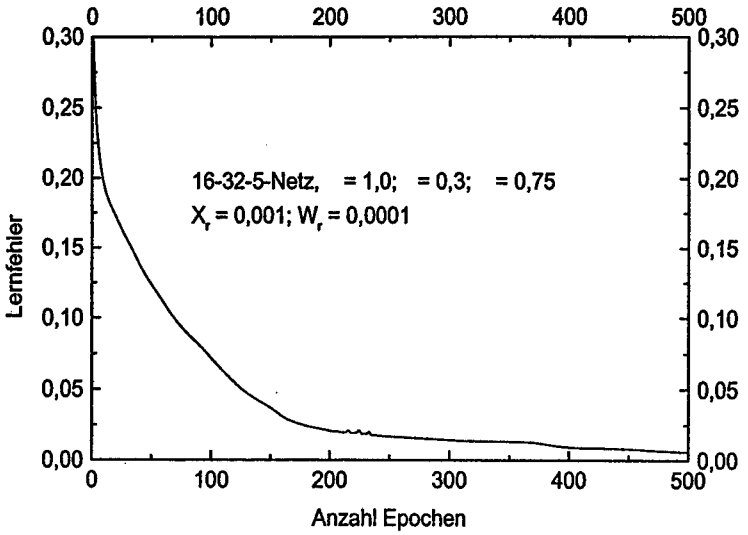


Abb. 4.42: *Ein traditionelles BP–Netz für das Beispiel* BEN_{52}

dem man die Anwendbarkeit des Verfahrens quantitativ abschätzen kann. Je größer $\theta_{\vec{x}}(0,1)$ ist, desto weiter liegen die Komponenten der Vektoren vom Rand des Intervalls [0,1] entfernt. Folgende Zuordnung ist eine triviale Lernaufgabe:

Eingangsvektor	Ausgangsvektor
(0.2 0.5 0.95)	0.5
(0.95 0.2 0.5)	0.2
(0.5 0.2 0.9)	0.9

die mit einem traditionellen BP–Algorithmus leicht gelöst werden kann (siehe Abbildung 4.43). Dagegen hat das Training mit dem vorgestellten Verfahren trotz der einfachen Aufgabe keinen Erfolg. Der Grund zum Mißerfolg ist leicht zu finden: $\theta_{\vec{x}}(0,1)$ ist für alle Eingangsvektoren gleich Eins.

Das Trainingsergebnis mit dem XOR–Beispiel liefert Konvergenz, solange $\theta_{\vec{x}}(0,1)$ aller vier Mustervektoren kleiner als 0,5 ist. Ob dieser Ansatz auf alle Aufgaben verallgemeinert werden kann, konnte aus Zeitgründen noch nicht studiert und bestätigt werden. Andererseits ist $\theta_{\vec{x}}(0,1)$ nur ein Maß für jeden einzelnen Mustervektor. Betrachtet man die gesamte Trainingsmenge als Ganzes, muß ein anderes vernünftiges Maß eingeführt werden.

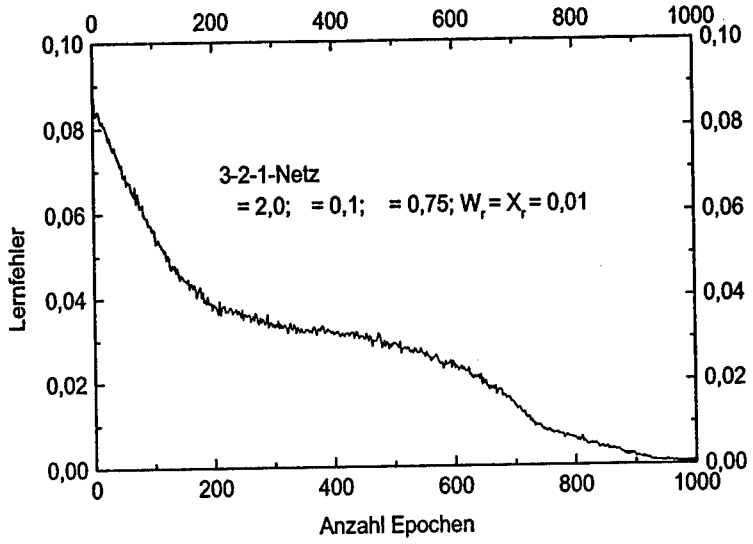


Abb. 4.43: Lernvorgang eines traditionellen BP–Netzes für ein triviales Beispiel

Der Begriff **Maximale Abweichung** ist für binäre Fälle ebenfalls nützlich, weil die Komponenten der Trainingsvektoren wegen stochastischer Streuungen nicht genau gleich dem Wert Eins oder Null sind. In dem Fall könnte ein Trainingsvektor immerhin aus dem Datensatz herausgenommen werden, wenn seine **Maximale Abweichung** $\theta_{\vec{x}}(0,1)$ wesentlich zu groß ist.

Aus den Ergebnissen der obigen Software–Simulationen lassen sich einige Schlußfolgerungen über die Eigenschaften des Verfahrens ziehen, die im nachstehenden Abschnitt detailliert besprochen werden. Zu diesem Zeitpunkt soll einiges über die Wirkungen des Einsatzes des ADDIE als Tiefpaßfilter (TPF) gesagt werden. Die Verwendung des ADDIE als TPF gegen die starke stochastische Streuung hat in der Praxis kaum positive Wirkungen gehabt, obwohl es bei dem XOR–Beispiel Vorteile aufgewiesen hat. Der Grund ist die Anlaufzeit, die das ADDIE braucht, um sich bei einem Musterwechsel am Netzeingang an die neue Neuronaktivität innerhalb des Netzes anzupassen. Wenn ein neuer Mustervektor am Netzeingang angelegt wird, müssen alle Neuronen im Netz darauf reagieren. Dies führt dazu, daß ihre Aktivitäten nach oben oder unten springen. Diese Änderung kann nicht kontinuierlich stattfinden. Andererseits ist die TPF–Wirkung des ADDIE nur bei langzeitiger Beobachtung sichtbar. Beim Training findet der Wech-

sel des Mustervektors meistens innerhalb der Größenordnung von einigen tausend Takten statt. Demzufolge pendelt der Wert des ADDIE zwischen zwei sehr unterschiedlichen Werten (z.B. Null und Eins) herum und erreicht nie einen stabilen Zustand. Daher kann es auf das Training auch nur einen negativen Einfluß ausüben, obwohl es die stochastischen Streuungen zu unterdrücken vermag.

4.5 Schlußbemerkung

Die vorgestellten Ergebnisse, die durch Software–Simulationen mit obigen Beispielen erhalten wurden, lassen sich zu folgenden Punkten zusammenfassen, welche die Schlußfolgerungen aus den in der vorliegenden Arbeit durchgeführten Untersuchungen darstellen:

- Das Gegenstrom–Verfahren, das vor dieser Arbeit als mögliche Ursache der Divergenz des Trainings angesehen wurde, ließ keinerlei negative Einflüsse auf die Konvergenzeigenschaften des vorgestellten Verfahrens erkennen. Außerdem hat die Verwendung des Gegenstrom–Verfahrens gegenüber dem *Online–Training* kaum nennenswerte Geschwindigkeitsunterschiede bei der Konvergenz gebracht. Dies ist der Befund aus Software–Simulationen mit zahlreichen Beispielen. Ob dies Allgemeingültigkeit hat, benötigt jedoch weitere theoretische Analysen in der Zukunft.

- Die Einführung des modifizierten Neurons hat gegenüber seinem Vorgänger deutliche Verbesserungen der Konvergenz gebracht. Dies ist sowohl bei seiner theoretischen Analyse als auch bei den Software–Simulationen mit zahlreichen Beispielen bestätigt worden. Erstens haben viele Beispiele nun konvergiert, die vorher unter Verwendung des alten Neurons in die Divergenz geraten sind. Zweitens ist die Auswahl der Trainingsparameter unter den neuen Neuronen viel flexibler als zuvor. Mit anderen Worten ist ein erfolgreiches Training nicht so stark von der richtigen Auswahl der Trainingsparameter abhängig wie zuvor. Dies ist besonders für die Hardware–Implementierung des Verfahrens geeignet, weil solche Parameter wie Zählerlänge von INDIE und ADDIE, $R-n$, δ usw. auf allgemeine geltende Werte in gewissen Grenzen festgelegt werden können. Offensichtlich dürfte das modifizierte Neuron nicht mehr Chipfläche benötigen als sein Vorgänger, weil der wesentliche Teil ein Schieberegister darstellt und der Komparator dadurch weggelassen werden kann, indem das Schieberegister mit einem gewissen Wert (je nach der Größe der Schwelle R_t) vorgeladen wird und das Überlauf–Bit des Schieberegisters als Ausgang

des Komparators dient. Der genaue Platzbedarf des modifizierten Neurons kann allerdings nur durch den Entwurf eines entsprechenden Prototypschaltkreises ermittelt werden.

- Das verwendete $\frac{1}{N}$–Mittelungsverfahren verändert die Konvergenzeigenschaft des Verfahrens nicht, sofern eine steilere S–Funktion zur Kompensation seines Einflusses genutzt wird. Dasselbe gilt ebenfalls für die $[0,1]$–Einschränkung. Bei gleichzeitigem Auftreten der beiden Einschränkungen ist die Verwendung einer sehr steilen S–Funktion erforderlich. Die neue S–Funktion kann diese Forderung besser erfüllen als ihre Vorgänger. Wenn die alte S–Funktion sehr steil wird, verhält sie sich fast wie eine Stufenfunktion und ihre Ausgangsfolge hat sehr schlechte statistische Eigenschaften gegenüber einer echte Zufallsfolge. Außerdem hat die B–Funktion, die als Vertreterin der Ableitung der S–Funktion dient, keinen logischen Zusammenhang mit der Steilheit der S–Funktion. Demzufolge kann die Abstiegsrichtung falsch festgestellt werden und dadurch kann das Training langsamer oder gar nicht konvergieren. Dagegen bleibt die neue S–Funktion unter starker Steilheit immer gut in S–Form und statt der B–Funktion wird eine geschätzte Ableitung der neuen S–Funktion für das Berechnen der Suchrichtung verwendet, die eine deutliche mathematische Beziehung zur Steilheit der neuen S–Funktion hat, d.h. die geschätzte Ableitung variiert mit der Änderung der Steilheit der neuen S–Funktion. Demzufolge kann eine angemessenere und genauere Abstiegsrichtung berechnet werden. Dies kann die Erklärung sein, warum die Einführung des modifizierten Neurons die Konvergenzeigenschaften des Verfahrens verbessern kann.

- Die Skalierbarkeit ist nicht von dem Verfahren selbst, sondern vom Schwierigkeitsgrad der zu lösenden Probleme abhängig. Aus diesem Grund fällt es schwer, die Skalierbarkeit des vorliegenden Verfahrens unabhängig von den zu lösenden Aufgabenstellungen vorherzusagen. Die Ergebnisse der obigen Simulationen mit unterschiedlichen Beispielen zeigen, daß sich ein Netz mit einer Größenordnung von knappen dreitausend Gewichten (Beispiel $BE_{10*8}(7)$) erfolgreich trainieren läßt. Die weitere Steigerung der Netzgröße bis zu etwa 10K Gewichten ist absehbar möglich. Dadurch kann qualitativ die Schlußfolgerung gezogen werden, daß eine Vergrößerung des Netzes kein prinzipielles Hindernis für einen erfolgreichen Einsatz des Verfahrens bildet. Dagegen ist der Schwierigkeitsgrad des zu lösenden Problems ein entscheidender Faktor für die Konvergenz des Verfahrens. Bei manchen Aufgabenstellungen kann sogar das Training eines kleinen Netzes oh-

ne Erfolg bleiben. In dem Sinne ist eine allgemeine Aussage über die Skalierbarkeit des Verfahrens nicht möglich. Für diejenigen Aufgabenstellungen, die das Verfahren behandeln kann, ist jedoch eine Aussage der Skalierbarkeit absehbar möglich. Deshalb ist es besser, zu fragen, mit welchen Problemklassen das vorgestellte Verfahren gut umgehen kann.

- Angesichts des Mechanismus des Verfahrens und der Ergebnisse der Software–Simulationen könnte allgemein behauptet werden, daß das vorgestellte Verfahren nur für binäre Aufgabenstellung geeignet ist. Dies kann als eine notwendige Voraussetzung für die Konvergenz des Trainings mit dem Verfahren betrachtet werden. In dieser Arbeit wurden viele Software–Simulationen mit unterschiedlichen nichtbinären Beispielen durchgeführt und keine davon hat einen Trend zur Konvergenz gezeigt. Sogar bei einer einfachen Approximation einer normalen Gauß–Funktion ist das Training in die Divergenz geraten. Dies deutet an, daß das Verfahren nicht in der Lage ist, mit nichtbinären Aufgabenstellungen umzugehen. Es muß jedoch betont werden, daß die Komponenten der Trainingsvektoren einer binären Aufgabenstellung nicht nur Nullen und Einsen sein müssen, solange ihre Werte in der Nähe von Null oder Eins liegen. Genauer gesagt, könnte das Training doch noch konvergieren, wenn die sogenannte **Maximale Abweichung** des Vektors $\vec{x}$ vom Wertbereichsrand Null oder Eins, d.h. $\theta_{\vec{x}}(0,1)$ kleiner als 0,5 ist. Die binäre Aufgabenstellung ist auch nicht unbedingt eine hinreichende Voraussetzung, weil bei einigen binären Beispielen das Training nicht immer konvergiert. Es ist aufgefallen, daß lediglich solche Minimumstellen durch das Verfahren erreicht werden können, die am Rand des gesamten Gewichtsraums liegen. Eine mögliche Erklärung wäre, daß die Werte der Gewichte in der Nähe des Ursprungs wegen starker stochastischer Streuungen nie stabil bleiben und somit keine stabilen Zustände hergestellt werden können. Für die Richtigkeit dieser Aussage muß eine weitere umfangreiche theoretische Analyse durchgeführt werden. Bis dahin gilt dies lediglich als eine Vermutung.

5 Zusammenfassung und Perspektiven

5.1 Zusammenfassung

Wie schon in der Einleitung beschrieben wurde, könnte der Einsatz von stochastischen Rechenwerken eine potentielle Lösung und ein interessantes Forschungsgebiet für die digitale Implementierung lernfähiger Neurochips sein. Der Ausfall der konventionellen Multiplikation kann Chipfläche einsparen und die unscharfe Datenverarbeitung kann den Trainingsvorgang des Netzes beschleunigen, weil das Training wegen des Bestehens der stochastischen Streuungen in diesem Fall nicht so leicht in einem lokalen Minimum steckenbleibt wie bei dem traditionellen BP–Algorithmus.

Den Ausgangspunkt der Arbeit bilden die Resultate von Riemschneider[51], der u.a. eine auf stochastischen Rechenwerken basierende parallele Hardware für Backpropagation–Netze implementiert hat. Die dort vorgestellten kaskadierbaren dezentralen Zufallsgeneratoren, speichernden Glieder (INDIE und ADDIE), Nichtlinearitäten, sowie deren Trainingsmechanismus, bei dem die Lern– und Arbeitsphase gleichzeitig auftreten (das sogenannte Gegenstrom–Verfahren), wurden hier übernommen und weiter untersucht.

Das Ziel der Arbeit war es, das in [51] vorgestellte Verfahren einer erweiterten wissenschaftlichen Analyse, Bewertung und einer möglichen Verbesserung zu unterziehen. Insbesondere wurden Aussagen über die Auswahl der Netzparameter, die Wirkungen des Gegenstrom–Verfahrens sowie der anderen einschränkenden Einflußgrößen, und vor allem die Abschätzung über die Skalierbarkeit und Anwendbarkeit des Verfahrens durch zahlreiche Beispiele mit Software–Simulationen gemacht. Der Weg zu diesem Ziel hat sich in folgende Schritten aufteilen lassen:

In der ersten Stufe der Arbeit wurde das vorgestellte Verfahren mit dem konventionellen BP–Algorithmus in Beziehung gesetzt. Dadurch wurde eine 1:1–Abbildung der Trainingsparameter zwischen beiden Algorithmen hergestellt. Daraus wurde die Unter– und Obergrenzen der Trainingsparameter mathematisch hergeleitet, welche für die Auswahl der Trainingsparameter als Leitfaden dienen können.

Die zweite Stufe behandelte die Analyse des Unterschieds zwischen dem vorgestellten Verfahren und dem konventionellen BP–Algorithmus. Das führte zu einer Liste potentiell einschränkender Einflußgrößen, deren Auswirkungen auf die Konvergenzeigenschaften des vorgestellten Verfahrens ausführlich untersucht wurden. Danach wurden mögliche Maßnahmen gegen

die Nebenwirkungen der Einschränkungen ergriffen und die entsprechende Technik zur Verbesserung des Verfahrens erläutert und softwaretechnisch für die Simulation implementiert. Mit geeigneten Beispielen wurden die aus der theoretischen Herleitung gewonnenen Aussagen durch Software–Simulation bestätigt.
In der letzten Stufe wurde ein Simulator für die Software–Simulation des ganzen Netzes aufgebaut, welcher das vorgestellte Verfahren taktgenau und hardwarenah simuliert und mit der objektorientierten Programmiersprache C++ geschrieben wurde. Der Simulator hat ebenfalls eine fensterorientierte Benutzeroberfläche und ist damit leicht zu bedienen. Jedoch ist die Software–Simulation für große Netze sehr zeitaufwendig und kann sogar Tage oder Wochen dauern. Daher ist es wichtig, die passenden Problemklassen für die Software–Simulation zu wählen. Aus diesem Grund wurden zunächst die zu behandelnden Problemklassen besprochen. Mit den Ergebnissen der Software–Simulation wurden dann einige wichtige Schlußfolgerungen gezogen, die der Verbesserung und Erweiterung des Verfahrens und eines möglichen Einsatzes in absehbaren Anwendungen als ein Wegweiser dienen können.

5.2 Perspektiven

Das hier untersuchte Verfahren zählt zur Art der sogenannten *Pulsed Neural Networks*. Im Gegensatz zu konventionellen ANNs, in denen die zu verarbeitenden Informationen durch kontinuierliche Datengrößen repräsentiert und zwischen Verarbeitungseinheiten (Neuronen) übertragen werden, verwenden *Pulsed Neural Networks* diskontinuierliche Datengrößen, nämlich Pulse, um Informationen darzustellen und zu verarbeiten. Neue Forschungsergebnisse aus dem Gebiet der biologischen neuronalen Netze haben aufgezeigt, daß dies genau die Arbeitsweise eines biologischen Neuro–Systems ist. Diese Tatsache regt viele Wissenschaftler zur Forschung auf diesem Gebiet an [40]. Jedoch findet man sehr wenig über die auf stochastischen Bitströmen basierenden neuronalen Netze und ihre Hardware–Implementierung in der Literatur, obwohl diese zu einer Untergruppe der *Pulsed Neural Networks* gehören. Dadurch bestehen noch viele offene Fragen auf diesem Forschungsgebiet, die vielleicht noch mehr Wissenschaftler aus vielen Disziplinen anregen könnten. Daher besteht kein Zweifel daran, daß noch viele offene Fragen nach dem Abschluß der vorliegenden Arbeit bleiben, die in der Zukunft weiter untersucht werden sollen.
So sind beispielsweise Untersuchungen der tatsächlichen Platzbelegung der modifizierten Neuronen erforderlich, wenn sie hardwaremäßig implementiert werden. Bisher wurde nur eine sehr grobe Abschätzung durch Vergleich

mit dem Blockschema der alten Vorgänger durchgeführt. Der Platzbedarf derartiger Neuronen auf dem Chip ist ein sehr wichtiger Faktor für die Beurteilung, ob die vorgeschlagenen modifizierten Neuronen außer ihren verbesserten Eigenschaften hardwaremäßig ebenfalls vorteilhaft sind.

Bisher konnte eine Aussage zur Lösbarkeit der Aufgabenstellung nur durch eine begrenzte Anzahl von Beispielen getroffen werden. Außerdem ist die Aussage, daß nur binäre Aufgaben für das Verfahren lösbar sind, keine hinreichende Bedingung, d.h. das auf dem vorgestellten Verfahren beruhende Netz muß sich nicht bei irgendeiner binären Aufgabe zwangsläufig mit Erfolg trainieren lassen. Es wäre ideal, wenn eine hinreichende Bedingung für die Lösbarkeit einer Problemklasse theoretisch hergeleitet werden könnte. Aber dies ist keine leichte Aufgabe, weil sich die Abbildung, die ein ANN repräsentiert, meistens nicht analytisch darstellen läßt. Ein anderer Weg ist es, durch zahlreiche Software–Simulationen und die daraus erhaltenen umfangreichen Ergebnissen eine statistische Aussage darüber zu erhalten. Dafür ist eine Verbesserung des bisherigen Softwaresimulators notwendig. Diese Verbesserung könnte sich in zwei Richtungen bewegen. Erstens könnte die Benutzer–Schnittstelle verbessert werden, so daß die Gewichtsänderungen und Ausgaben jedes Neurons während des Trainings quasi in Echtzeit grafisch dargestellt werden können. Damit kann man den Trainingsvorgang besser verfolgen und die Werte von bestimmten Gewichten beim Training nach Bedarf manuell ändern. Dies bietet die Möglichkeit, das Training aus einem lokalen Minimum schnell herauszuführen. Zweitens besteht die Möglichkeit der Parallelisierung des Simulators. Dies ist eigentlich eine intrinsische Eigenschaft eines neuronalen Netzes, weil die Informationsverarbeitung nur lokal stattfindet. Wenn die Simulation des Verfahrens auf einem Multiprozessor–Rechner echt parallel laufen kann, wird der Zeitaufwand für die Simulation eines relativ großen Netzes deutlich abnehmen.

Über die Skalierbarkeit des vorliegenden Verfahrens konnte bisher ebenfalls nur eine sehr grobe Abschätzung durch Simulation mit begrenzten Beispielen getroffen werden. Mit Hilfe des Gesetzes der großen Zahl aus der Theorie der Wahrscheinlichkeit und Statistik (siehe Anhang B) könnte möglicherweise eine quantitative Aussage theoretisch hergeleitet werden. Der Ausgangspunkt besteht darin, daß die Eingangsgröße eines Neurons als eine Zufallsgröße X_i und alle Eingänge des betroffenen Neurons als eine Folge $\{X_i\}$ von Zufallsgrößen betrachtet werden. Ein zu X_i gehöriges Elementarereignis ist somit ein reeller Wert, der im Interval [0,1] liegt. Er wird von dem gerade am Netzeingang angelegten Trainingsmuster bestimmt. Daher ist die Anzahl der zur Zufallsgröße X_i gehörigen Elementarereignisse von der Anzahl K der zur Verfügung stehenden Trainingsmuster abhängig. In diesem Fall wird die Anzahl der Zufallsgrößen in der Folge $\{X_i\}$ gleich der

Anzahl n der Eingänge eines Neurons. Nun kann man mit Hilfe des Gesetzes der großen Zahl eine Obergrenze für n herleiten. Wird die Obergrenze der Eingänge eines Neurons gefunden, kann eine Aussage über die Obergrenze des gesamten Netzes getroffen werden.

Wegen der Nebenwirkung seiner Anlaufzeit hat die Verwendung des TPF zum Unterdrücken der Streuungen keinen signifikanten Gewinn gebracht. Ob noch andere Techniken für diesen Zweck bestehen, ist ebenfalls eine Untersuchung zur Verbesserung des Verfahrens wert. Die starke stochastische Streuung in der Nähe des Ursprungs verfälscht die Suchrichtung des Abstiegs meistens, besonders wenn die S–Funktion sehr steil ist. Es könnte eine Lösung sein, andere bekannte Filter aus dem Gebiet der Elektrotechnik auszuprobieren, soweit sie zum Verfahrensmechanismus passen.

Literaturverzeichnis

[1] Bryson, A., Ho, Y.-C. *'Applied Optimal Control'*. Blaisdell, New York, 1969.

[2] Balakrishnan, K., Honavar, V. *'Improving convergence of backpropagation by handling flat spots in the output layer'*. In: 'Artificial Neural Networks', Amsterdam, North Holland, 1992.

[3] Bishop, C. *'Neural Networks for Pattern Recognition'*. Oxford University Press, Oxford, London, 1995.

[4] Brandt, H. *'Untersuchungen zur Anwendbarkeit stochastischer Rechenverfahren in Neuronalen Netzen'*. Technischer Bericht, Technische Informatik, Universität der Bundeswehr Hamburg, 1993.

[5] Crockett, F., Garner, J. *'MFC Entwickler Workshop'*. Microsoft Press Deutschland, Unterschleißheim, 1997.

[6] Cichocki, A., Unbehauen, R. *'Neural Networks for Optimization and Signal Processing'*. Wiley Verlag, Stuttgart, 1993.

[7] Dickson, J. *'Stochastic Arithmetic Implementations of Neural Networks with In Situ Learning'*. Proceedings of ICNN, Vol. 2, 1993.

[8] Erten, G., Goodman, R. M. *'A Digital Neural Network Architecture Using Random Pulse Trains'*. In: 'Proceedings of IEEE ICNN', Band 1, Department of Electrical Engineering,116-81, California Institute of Technology, Pasadena, CA 91125, 1992.

[9] Fahlmann, S. *'Faster-learning variations on back-propagation'*. In: 'Proceedings of the 88' Connectionist Models Summer School', Carnegie-Mellon-University, 1988.

[10] Funahashi, K. *'On the approximate realization of continuous mappings by neural networks'*. Neural Networks, 2:192–193, 1989.

[11] Graf, H., deVegar, P. *'Advanved Research on VLSI'*. MIT Press, Cambridge,MA, 1987.

[12] Graf, H., DeVegar, P. *'A CMOS associative memory chip'*. In: 'Proceedings of IEEE ICNN', San Diego, 1987.

[13] Gurewich, N., Gurewich, O. *'Visual C++ in 21 Tagen'*. SAMS, 1996.

[14] Grosche, G., V.Ziegler. *'Taschenbuch der Mathematik'*. Teubner Verlagsgesellschaft, Leipzig, 1979.

[15] Härtter, E. *'Wahrscheinlichkeitsrechnung, Statistik und mathematische Grundlagen'*. Vandenhoeck und Ruprecht, 1987.

[16] Hein, R. *'Untersuchungen zu Gradientenmethoden in neuronalen Netzen mit stochastischer Arithmetik'*. Technischer Bericht, Professur Technische Informatik,Universität der Bundeswehr Hamburg, 1994.

[17] Hikawa, H. *'Implementation of Simplified Multilayer Neural Networks with On-chip Learning'*. In: 'Proceedings of IEEE ICNN', Oita University, Japan, 1995.

[18] HLAVACKOVA, K., KURKOVA, V. *'Rates of Approximation of Real-Valued Boolean Functions by Neural Networks'.* In: 'Proceedings of ESANN 96, Bruges'. Acad. of Sciences (Czech Republic), 1996.

[19] HOFFMANN, N. *'Simulation neuronaler Netze: Grundlagen, Modelle, Programme'.* Vieweg, Braunschweig, 1991.

[20] HUSH, D., SALAS, J. *'Improving the learning rate of back-propagation with the gradient reuse algorithm'.* In: 'Proceedings of the IEEE 1st International Conference on Neural Networks', 1987.

[21] HORNIK, K., STINCHCOMBE, M. *'Multilayer feedforward networks are universal approximators'.* Neural Networks, 2, 1989.

[22] JACOBY, S., KOWALIK, J., PIZZO, J. *'Iterative Methods for Nonlinear Optimization Problems'.* Prentice-Hall Inc., 1972.

[23] JOU, I., TSAY, Y. *'Parallel Distributed Processing with Multiple One-Output Back-Propagation Neural Networks'.* In: 'International Symposium on Circuits and Systems', Singapore, 1991.

[24] KARHUNEN, J. *'Neural approaches to independent component analysis and source separation'.* In: 'Proceedings of ESANN 96, Bruges'. Helsinki University of Technology, Finland, 1996.

[25] KASPRZAK, W., CICHOCKI, A. *'Recurrent least square learning for quasi-parallel principal component analysis'.* In: 'Proceedings of ESANN 96, Bruges'. RIKEN Frontier Rearch(Japan), 1996.

[26] KIM, I. *'Multilayer Network with Bipolar Weights'.* In: 'Proceedings of IEEE ICNN'. GoldStar Central Research Laboratory, 1994.

[27] KNOTHE, R. *'VHDL-Beschreibung von Neuro-Hardware mit stochastischen Rechenwerken'.* Technischer Bericht, Laborarium für Elektronik und Nachrichtenverarbeitung, Universität der Bundeswehr Hamburg, 1996.

[28] KRATZER, K. *'Neuronale Netze: Grundlagen und Anwendungen'.* Carl Hanser Verlag, 1991.

[29] KREYSZIG, E. *'Statistische Methoden und ihre Anwendungen'.* Vandenhoeck und Ruprecht, Göttingen, 1968.

[30] KUNG, S. *'Digital Neural Networks'.* PTR Prentice Hall, Englewood Cliffs, New Jersey 07632, 1993.

[31] KARAYIANNIS, N. B., VENETSANOPOULOS, A. N. *'Artificial Neural Networks: Learning Algorithms, Performance Evaluation, and Application'.* Kluwer Academic Publishers, Boston/Dordrecht/London, 1993.

[32] LAU, C. *'Neural Networks: Theoretical Foundations and Analysis'.* IEEE Press, New York, 1992.

[33] LAHTOKANGAS, M., KORPISAARI, P. *'Maximum covariance method for weight initialization of multilayer perceptron network'.* In: 'Proceedings of ESANN 96, Bruges'. * Nokia Research Center, ** Tampere Univ. of Tech. (Finland), 1996.

[34] MAUNDY, B., ABD EL-MASRY, E. *'Switched-capacitor neural networks using pulse based arithmetic'.* Electronics Lett., 26 (15), 6 1990.

[35] MALLOFRE, A. C. *'A fast Bayesian algorithm for Boolean functions synthesis by means of perceptron networks'.* In: 'Proceedings of ESANN 96, Bruges'. Univ. Polit. Catalunya (Spain), LAAS-CNRS(France), 1996.

[36] MASSEN, R. *'Stochastische Rechentechnik'.* Carl Hanser Verlag, 1977.

[37] MASTERS, T. *'Practical neural network recipes in C++'.* Academic Press, Inc., 1993.

[38] MASTERS, T. *'Signal and Image Processing with Neural Networks: A C++ Sourcebook'.* Wiley, New York, USA, 1994.

[39] MAYORAZ, E. *'Bounds on the Degree of High Order Binary Perceptrons'.* In: 'Proceedings of ESANN 96, Bruges'. IDIAP(Switzerland), 1996.

[40] MAASS, W., BISHOP, C. M. *'Pulsed Neural Networks'.* MIT Press, 1998.

[41] MARCHESI, M., ORLANDI, G. *'Fast Neural Networks Without Multipliers'.* IEEE Trans. on Neural Networks, 4 (1), 1993.

[42] MINSKY, M., PAPERT, S. *'An Introduction to Computational Geometry'.* MIT Press, 1969.

[43] MICHIE, D., SPIEGELHALTER, D., TAYLOR, C. *'Machine Learning, Neural and Statistical Classification'.* Ellis Horwood, 1994.

[44] MURRAY, A., TARASSENKO, L. *'Analogue Neural VLSI—A pulse stream approach'.* Chapman and Hall, 1994.

[45] NAEVE, P. *'Stochastik für Informatik'.* Oldenbourg, München, 1995.

[46] NGUYEN, D., HOLT, F. *'Stochastic Processing in a Neural Network Application'.* In: 'Proceedings of IEEE ICNN, Vol. 3', Bellevue, WA, USA, 1987. Boeing High Tech Center.

[47] OSSEN, A., RÜGER, S. *'An analysis of the metric structure of the weight space of feedforward networks and its application to time series modeling and prediction'.* In: 'Proceedings of ESANN 96, Bruges'. TU Berlin, 1996.

[48] PFISTER, M., ROJAS, R. *'Backpropagation Algorithms: Their Characteristics and Efficiency'.* Technischer Bericht, FU Berlin, Fachbereich Mathematik, Institut für Informatik, 1993.

[49] PRECHELT, L. *'PROBEN 1-A Set of Neural Network Benchmark Problems and Benchmarking Rules'.* Technischer Bericht 21/94, Fakultät für Informatik, Universität Karlsruhe, Germany, September 1994.

[50] RUMELHART, D., HINTON, G. *'Learning representations by back-propagating errors'.* Nature, 323, 1986.

[51] RIEMSCHNEIDER, K.-R. *'Parallele Hardware für Backpropagation–Netze auf der Basis stochastischer Rechenwerke'.* Dissertation, UniBw Hamburg, 1996.

[52] RITTER, H. *'Neuronale Netze: Eine Einführung in die Neuroinformatik selbstorganisierender Netzwerke'.* Addison-Wesley, Bonn, 1991.

[53] ROGER, K., KINSNER, W., MCLEOD, R. *'A Stochastic Arithmetic Back-propagation Neural Network'.* Technischer Bericht, Department of Electrical and Computer Engineering, Uni. of Manitoba, CA, 8 1993.

[54] ROGER, K. *'FPGAs Implementation of a Stochastic Arithmetic Neural Network'.* In: 'Proceedings of CMC Workshop On FPGAs'. Department of Electrical and Computer Engineering, University of Manitoba, CA, 1994.

[55] ROJAS, R. *'Theorie der neuronalen Netze: Eine systematische Einführung'.* Springer–Lehrbuch, 1993.

[56] RÜBEL, M., 1998. UniBw Hamburg, Lab. für Elektronik und Nachrichtenverarbeitung, Persönliche Mitteilung.

[57] RUMELHART, D. E. *'Parallel distributed processing'.* The MIT Press, London, 1987.

[58] SALOMON, R. *'Verbesserung konnektionistischer Lernverfahren, die nach der Gradientenmethode arbeiten'.* Dissertation, TU Berlin, 11 1992.

[59] SCHÖNEBURG, E. *'Industrielle Anwendung Neuronaler Netze'.* Addison-Wesley, Bonn, 1993.

[60] STORNETTA, W., HUBERMANN, B. *'An improved three–layer backpropagation algorithm'.* In: 'Proceedings of the IEEE 1st International Conference on Neural Networks', 1987.

[61] SLATE, D. J. *'Letter Recognition Using Holland-Style Adaptive Classifiers'.* Machine Learning, 6 (2), March 1991.

[62] SÁNCHEZ-SINENCIO, E., LAU, C. *'Artificial Neural Networks: Paradigms, Applications and Hardware Implementations'.* IEEE Press, New York, 1992.

[63] SHAWE-TAYLOR, J., JEAVONS, P., VAN DAALEN, M. *'Probabilistic Bit Stream Neural Chip: Theory'.* Connection Science, 3 (3), 1993.

[64] SUTTON, R. *'Reinforcement Learning: An Introduction'.* The MIT Press, 1998.

[65] TOMLINSON JR, M. S., WALKER, D. *'A Digital Neural Network Architecture for VLSI'.* In: 'Proceedings of IEEE IJCNN', Band 2, San Diego, 1990.

[66] VARFIS, A., CORLETO, L. *'Error rate estimation via cross-validation and learning curve theory'.* In: 'Proceedings of ESANN 96, Bruges'. CEC Joint research Center Ispra(Italy), 1996.

[67] VAN DAALEN, M., JEAVONS, P., SHAWE-TAYLOR, J. *'A stochastic neural architecture that exploits dynamically reconfigurable FPGAs'.* In: 'Proceedings of IEEE Workshops on FPGAs for Custom Computing Machines', Napa,CA, 1993.

[68] VAN DAALEN, M., J.ZHAO, SHAWE-TAYLOR, J. *'Real time output derivatives for on chip learning using digital stochastic bit stream neurons'.* Electron. Lett., 30 (21), 1994.

[69] VAN DAALEN, M., KOSEL, T., JEAVONS, P., SHAWE-TAYLOR, J. *'Emergent Activation functions from a stochastic Bit–Stream neuron'.* Electron. Lett., 30 (4), 1994.

[70] WEBER, H. *'Einführung in die Wahrscheinlichkeitsrechnung und Statistik für Ingenieure'.* B.G.Teubner, Stuttgart, 1992.

[71] WERBOS, P. *'Beyond Regression: New Tools for Prediction and Analysis in the Behavioral Science'.* Dissertation, Harvard University, 11 1974.

[72] XUDONG, J. *'Effiziente Mustererkennung durch spezielle neuronale Netze'.* Dissertation, UniBw Hamburg, 1997.

[73] ZIMMERMANN, H.-J. *'Datenanalyse'.* VDI Verlag, Düsseldorf, 1995.

[74] ZAGHLOUL, M., MEADOR, J., NEWCOMB, R. *'Silicon Implementation of Pulse Coded Neural Networks'.* Kluwer Academic, Boston/Dordrecht/London, 1994.

A Simulationssoftware

A.1 Aufbau des Programms

Der Software–Simulator, der den Hardware–Vorgang des Verfahrens taktgenau simuliert, wurde in der Microsoft–Visual–C++–Umgebung unter Anwendung der *Microsoft Foundation Classes (MFC)* erstellt. Dabei handelt es sich um eine fensterorientierte SDI–Anwendung (*Single Document Interface*[1]), welche das Training der Netze mit einer beliebigen Anzahl von Gewichten in der Ausgangsschicht und der verborgenen Schicht und unterschiedlichen Trainingsstrategien (z.B *Batch*, *Online*, Gegenstrom–Verfahren, Teil– oder Vollverbindung, usw) simulieren kann. Die Simulation des vorgeschlagenen Verfahrens durch Software ist sehr zeitaufwendig. Aus diesem Grund soll der Simulator in der Lage sein, die Zwischenergebnisse des Trainings jederzeit beobachten oder überprüfen sowie den Lernvorgang gegebenenfalls jederzeit unterbrechen zu können. Der Zeitaufwand kann damit optimal gestaltet werden. Aus diesem Grund wird die sogenannte *Multi–Threading*–Technik verwendet. Der Simulator hat während des Simulations–Durchlaufs zwei gleichzeitig laufende *Threads*, nämlich einen *Training–Thread* für die Ausführung des Verfahrens selbst und einen *Control–Thread* für die Überwachung des Lernvorgangs.

A.1.1 Zufallsquelle

Die Zufallsquelle spielt im vorgeschlagenen Verfahren die entscheidende Rolle, weil sie zahlreiche unabhängige Hilfsbitfolgen (hier ist von der 0,5–Folge die Rede), die für die Codierung der Maschinenvariablen und die korrekte Arbeitsweise des Verfahrens zuständig sind, gleichzeitig erzeugen muß. Weiterhin müssen die erzeugten Zufallsfolgen bestimmte Zufallseigenschaften besitzen. In [51] wurde eine geschickte Lösung gefunden, bei der Pseudozufallsquellen, nämlich Schieberegister mit und ohne Rückkopplung, verwendet werden. Diese Lösung ist besonders günstig für eine digitaltechnische Implementierung des vorgeschlagenen Verfahrens. Im Prototyp–Schaltkreis wurden von einem 28–Bit–Register 56 Folgen großer sowie drei Folgen geringerer Verschiebung abgeleitet (zu Einzelheiten siehe [51]). Durch M–fache

[1] Bei einer SDI–Anwendung kann zu einem bestimmten Zeitpunkt immer nur ein Rahmenfenster geöffnet sein. Entsprechend kann zu jedem Zeitpunkt nur ein neuronales Netz simuliert werden.

Kaskadierung solcher Register können knapp $60 * M$ Zufallsfolgen gleichzeitig erzeugt werden. Die erzeugten Folgen haben alle üblichen Zufallstests bestanden. Aus dieser Sicht dürfte die Anzahl der zu erzeugenden Zufallsfolgen für die Skalierbarkeit des Verfahrens kein Hindernis sein.

Wenn man den Durchlauf des Verfahrens taktgenau per Software simulieren will, kann der vom Compiler mitgelieferte Zufallsgenerator nicht verwendet werden, weil er nicht in der Lage ist, zahlreiche unabhängige Zufallsfolgen gleichzeitig zu erzeugen. So wird der oben erläuterte Zufallsgenerator für die Simulation per Software implementiert. Offensichtlich ist die Anzahl der Kaskadierungen der Schieberegister von der Größe des zu simulierenden Netzes abhängig und das Verteilen der erzeugten Zufallsbits an die „Verbraucher" ist von der Netzstruktur bestimmt. Mit einer selbst definierten Klasse unter C++ kann dies leicht realisiert werden.

Quelltext 5.1 Die Klasse des Zufallspools

```
class CZufallpool
{
int znnr;
int zufallindex;
BYTE * pZufall;

long sr(void);
void gor(long sr,BYTE *f,long *kaska);

public:
CZufallpool();
-CZufallpool();
BOOL init( int znn, CWnd *pCWnd );
void fillpool(void);
BYTE pob(void);
};
```

Die Ganzzahl *znnr* ist das Vielfache von 60, das von der gesamten Anzahl der benötigten Zufallsfolgen im Netz bestimmt wird. Nach dieser ganzen Zahl wird ein Zufallspool dynamisch eingerichtet, der bei jedem Takt mit Zufallsbits gefüllt werden soll und dessen Adresse über den Zeiger pZufall angesprochen wird. Wird ein Zufallsbit von einer Recheneinheit verlangt, kann sie durch die Methode pob ein Zufallsbit aus dem eingerichteten Zufallspool herausholen. Dazu wird am Anfang jedes Taktdurchlaufs der Zufallspool mit den von Schieberegisterketten erzeugten Zufallsbits neu gefüllt.

A.1.2 Neuronen und Synapsen

Neuronen und Synapsen sind Kernstücke eines biologischen neuronalen Netzes. Bei einem ANN gilt dies ebenfalls. Unabhängig davon, welche Größe oder Topologie ein ANN besitzt, besteht es immer aus solchen identischen und gleichartig fungierenden Kernstücken. Diese Eigenschaft paßt sehr gut zur objektorientierten Programmierungstechnik. Wenn ein ANN per Software nachgebildet werden soll, lassen sich zwei grundlegende Klassen definieren, nämlich Cneuron und Csynapse, die aus CObject, einer der vordefinierten Klassen der MFC, abgeleitet werden. Die objektorientierte Programmiermethode hat dabei den Vorteil, daß Daten und die dazu gehörigen Operationen (auch Methoden genannt) gekapselt werden. Damit ist ein Objekt einer bestimmten Klasse in der Lage, eine zugeordnete Aufgabe allein zu erledigen, z.B. Empfangen und Verarbeiten der Nachrichten von der Außenwelt, die Modifikation seines eigenen Zustands, usw. Aus dieser Sicht ist die objektorientierte Programmierungstechnik für die Simulation eines ANN besonders geeignet, weil ein ANN aus vielen identischen Objekten (Neuronen und Synapsen) besteht, welche die eigenen Aufgaben allein erledigen sollen.

Zur Untersuchung der Skalierbarkeit des vorgeschlagenen Verfahrens muß das Programm für die Simulation in Netzen beliebiger Größe einsetzbar sein. Dabei wird die in [51] durchgeführte Diskussion über die strukturelle Modellgestaltung eines Backpropagation–Netzes zur Anwendung gebracht. Gemäß der dort erläuterten S–A–N–Kategorisierung werden Synapsen, Akkumulationen und Neuronen zusammen als Verarbeitungseinheiten betrachtet. Für die Software–Simulation wird ein **Kombi** als eine Verarbeitungseinheit vorgeschlagen. Ein **Kombi** besteht aus einem Neuron und zahlreichen Synapsen, die dem Neuron eingangsseitig Informationen zuleiten. Die Abbildung A.1 zeigt das Funktionsschema eines Kombi:

In der Abbildung entsprechen n_{l-1} und n_{l+1} der Neuronenanzahl der vorstehenden Schicht oder der Eingangsdimension (falls $l = 1$), beziehungsweise der Neuronenanzahl der nachstehenden Schicht oder der Ausgangsdimension. Daher kann ein Backpropagation–Netz in Schichten und jede Schicht in Kombis zerlegt werden. Für eine solche Verarbeitungseinheit wird entsprechend eine Klasse CCombi eingeführt, die ebenfalls aus CObject abgeleitet wird. Die Klasse CCombi enthält einen Zeiger, der auf ein Element vom Typ Cneuron zeigt, und ein Feld von Elementen des Typs Csynapse. Die Größe des Csynapse–Feldes wird dynamisch belegt, je nach der Struktur des zu simulierenden Netzes und der Dimension der Trainingsvektoren. Daher

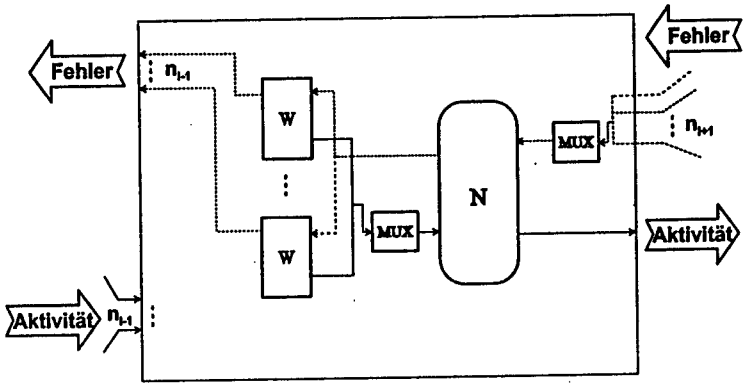


Abb. A.1: Funktionsschema eines Kombi

ist das Programm in der Lage, Backpropagation–Netze beliebiger Größe mit dem vorgeschlagenen Verfahren zu trainieren[2].

Von der vordefinierten MFC–Klasse CDocument wird die Klasse CBpteilnetzDoc abgeleitet, die zur Speicherung und Verwaltung der eigenen Daten der Simulation dient, z.B. Daten der Netz–Topologie, Trainingsparameter und gelernte Gewichte des Netzes. Das Kernstück in dieser Klasse sind zwei Member–Variablen, m_UnitArray und m_OutArray, welche die verborgene beziehungsweise die Ausgangsschicht des Netzes darstellen. Die beiden Variablen sind wiederum Felder von Objekten der Klasse CCombi. Durch das Überladen der Mitgliederfunktion Serialize der Klasse CDocument können die Daten einer Simulation jederzeit auf die Festplatte gespeichert und in den Hauptspeicher zurückgeladen werden.

A.2 Schnittstelle zum Benutzer und Ablauf des Programms

Das Simulationsprogramm ist eine SDI–Anwendung mit vielfachen Ansichten. Die Anwendung enthält zwar nur ein Rahmenfenster, jedoch können mehrere Ansichtenobjekte darin untergebracht werden. Durch Wählen auf der Menüleiste „Ansicht" kann man zwischen den Ansichten umschalten, um beispielsweise den Lernfehler während des Trainings grafisch anzeigen zu lassen oder die gelernten Gewichte und die Netzaktivität im Text–Modus

[2]Auf Grund der langen Trainingszeit kommen zur Zeit nur zweischichtige BP–Netze in Betracht. Es ist jedoch auch für Netze mit mehr Schichten einsetzbar.

anzuschauen (siehe Abbildung A.2). Dies hat den Vorteil, daß viele nützliche Informationen auf der begrenzten Anzeigenfläche untergebracht werden können. Die Anzeige des Lernfehlers während des Trainings geschieht *online*. Dadurch kann der Lernvorgang direkt beobachtet und zu jeder Zeit abgebrochen werden, wenn kein Konvergenztrend beim Training zu erkennen ist.

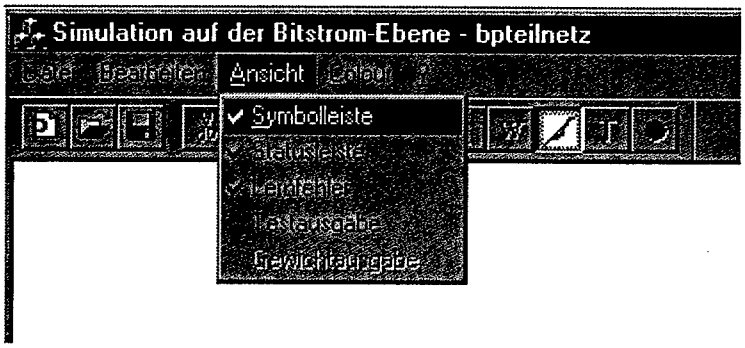


Abb. A.2: *Umschalten zwischen Ansichten*

A.2.1 Vorbereitung der Trainingsmuster–Datei

Wird vom Training des BP–Netzes gesprochen, muß eine Menge von Trainingsmustern vorhanden sein. Alle vorhandenen Trainingsmuster werden in einer Datei untergebracht, und zwar in folgendem Format:

$$
\begin{array}{cccccccc}
x_{11} & x_{12} & \cdots & x_{1n} & y_{11} & \cdots & y_{1m} \\
x_{21} & x_{22} & \cdots & x_{2n} & y_{21} & \cdots & y_{2m} \\
\multicolumn{7}{c}{\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots} \\
x_{p1} & x_{p2} & \cdots & x_{pn} & y_{p1} & \cdots & y_{pm}
\end{array}
$$

x_{ij} ist die Eingabe i des Musters j und y_{ij} die Ausgabe i des Musters j (i=1,...,n; j=1,...,m). n und m sind die Dimensionen der Eingabe- beziehungsweise Zielvektoren. Außer dem Leerzeichen und dem Tabulator werden keine anderen Zeichen als Trennzeichen zwischen Daten verwendet.

A.2.2 Programmablauf

Das Programm hat eine anwenderfreundliche, grafische Benutzeroberfläche, die man leicht bedienen kann. Durch Ausführen von bpteilnetz.exe wird das

Programm gestartet und eine fensterorientierte Oberfläche wie in Abbildung A.2 wird aufgeschlagen.
Durch Klicken des mit dem Buchstaben P gekennzeichneten Knopfes oder des Menüpunktes New unter Menü Datei wird ein Dialogfenster geöffnet (siehe Abbildung A.3). Dabei handelt es sich um ein **nichtmodales** Dialogfenster, d.h. das Dialogfenster kann im Gegensatz zu einem **modalen** Dialogfenster offen bleiben, wenn das Programm fortgesetzt wird (für Einzelheiten siehe [5] und [13]). Dies erweist sich als sinnvoll, wenn man während des Trainings die Trainingsparameter beobachten oder sogar verändern möchte. Im Dialogfenster kann die Topologie des zu simulierenden

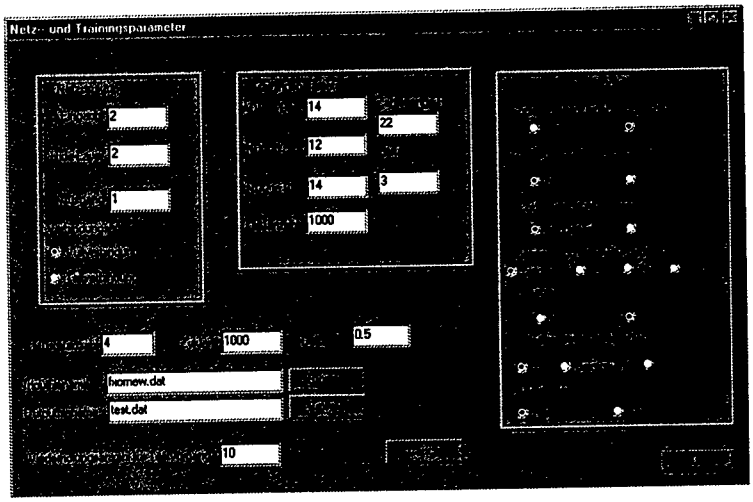


Abb. A.3: Dialogfenster für die Eingabe der Trainingsparameter

Netzes (Neuronenanzahl in der verborgenen Schicht und Dimensionen der Ein- beziehungsweise Ausgangsvektoren sowie Verbindungsart), die Werte der Lernparameter (ADDIE- und INDIE-Zählerlänge für Synapsenglieder, Taktanzahl, Runlänge der stochastischen Automaten für Neuronen oder die Schwellenwerte der modifizierten Neuronen) und Trainingsstil (Codiererart, Neuronenart, Reihenfolge des Anlegens von Trainingsmustern, Initialisierungsmethode für die Anfangswerte der Synapsen, Gegenstromverfahren oder Online-Training usw.) festgelegt werden. Darüber hinaus werden auch die Dateinamen der Trainingsvektoren und des Trainingsergebnisses und die Dauer des Trainings (Zyklen) angegeben. Erst nach der Bestätigung durch den OK-Knopf stehen die angegebenen Werte dem Programm zur Verfügung.

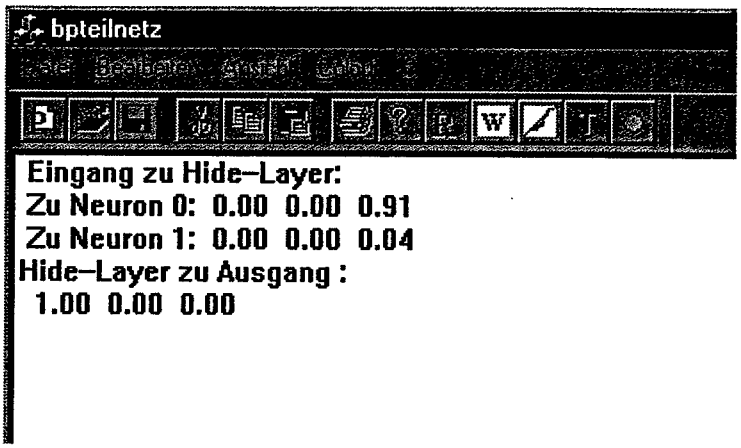


Abb. A.4: Anzeigen der Gewichte im Text–Format

Nach dem Klicken des R–Knopfes wird das Training des BP–Netzes gestartet. Danach kann man durch Auswählen der entsprechenden Menü–Items das Training anhalten und bei Bedarf wieder fortfahren. Durch Anklicken des W–Knopfes und seines rechten Nachbar–Knopfes kann man sich die gerade gelernten Gewichte im Text–Format oder den Lernfehler im grafischen Format anzeigen lassen (siehe Abbildungen A.4 und A.5).

Nach dem Abschluß des Trainings kann ein Test–Vorgang durch Anklicken des T–Knopfes gestartet werden. Der Test–Vorgang kann sowohl auf derselben Mustermenge, die für das Training verwendet wurde, als auch auf einer neuen Mustermenge, die dem Netz unbekannt ist, durchgeführt werden. Die Ausgaben des Netzes zu den entsprechenden Eingaben der Mustermenge werden im Text–Format angezeigt (siehe Abbildung A.6). Die Ausgaben sind die arithmetischen Mittelwerte der Netz–Ausgaben von mehrfachen Durchläufen des Test–Vorgangs.

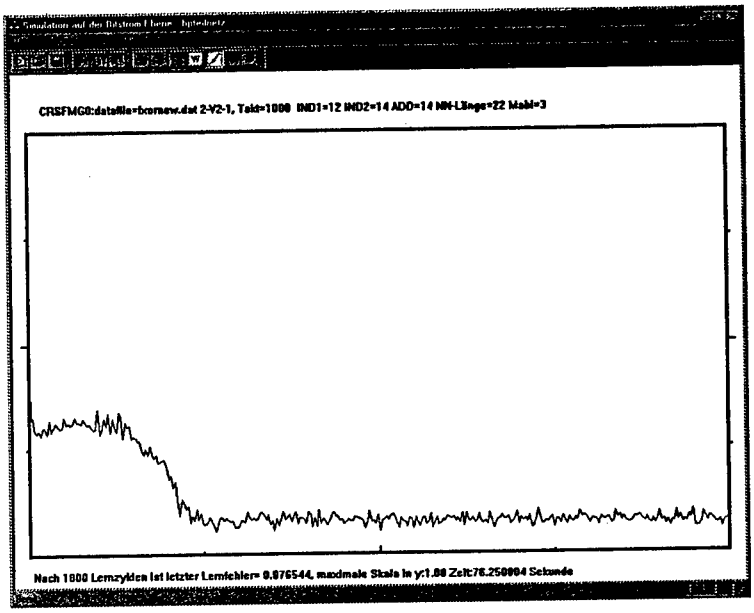


Abb. A.5: Anzeigen des Lernfehlers im grafischen Format

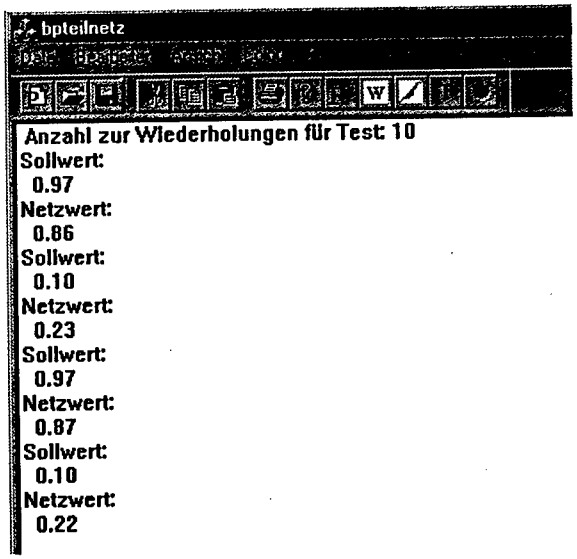


Abb. A.6: Anzeigen der Netz–Ausgaben beim Test–Vorgang

B Gesetze der großen Zahl und Skalierbarkeit

B.1 Gesetze der großen Zahl

Folgende Begriffe wurden aus [14] herangezogen:

Definition 1: Eine Folge$\{X_i\}$ von Zufallsgrößen heißt konvergent in Wahrscheinlichkeit gegen die Zufallsgröße X, wenn für beliebiges $\epsilon > 0$ gilt:

$$\lim_{i\to\infty} P(|X_i - X| < \epsilon) = 1 \tag{B.1}$$

Dabei bedeutet die Konvergenz in Wahrscheinlichkeit gegen die Zufallsgröße X, daß die Folge Folge$\{X_i\}$ im stochastischen Sinne gegen X konvergiert. Mit anderen Worten ist es im Grenzfall $i \to \infty$ also „fast sicher", daß sich X_i vom Grenzwert X beliebig wenig unterscheidet [70]. Es gibt noch ein stärkeres Konvergenzverhalten, das lautet:

Definition 2: Eine Folge $\{X_i\}$ von Zufallsgrößen konvergiert **mit Wahrscheinlichkeit 1** gegen X, wenn gilt

$$P(lim_{i\to\infty} X_i = X) = 1 \tag{B.2}$$

In diesem Fall kann man sagen, daß die Folge $\{X_i\}$ im stochastischen Sinne „sicher" gegen X konvergiert.

Definition 3: Man sagt, eine Folge X_1,... von Zufallsgrößen ist dem schwachen Gesetz der großen Zahl unterworfen, wenn für beliebiges $\epsilon > 0$ gilt:

$$\lim_{n\to\infty} P(|\frac{1}{n}\sum_{i=1}^{n} X_i - \frac{1}{n}\sum_{i=1}^{n} EX_i| < \epsilon) = 1 \tag{B.3}$$

EX_i ist der Erwartungswert der Zufallsgröße X_i und ist normalerweise eine Konstante.

Definition 4: Man sagt, eine Folge X_1,... von Zufallsgrößen ist dem starken Gesetz der großen Zahl unterworfen, wenn gilt:

$$P(\lim_{n\to\infty} (\frac{1}{n}\sum_{i=1}^{n} X_i - \frac{1}{n}\sum_{i=1}^{n} EX_i) = 0) = 1 \tag{B.4}$$

Wenn eine Folge X_1,... von Zufallsgrößen dem starken Gesetz der großen Zahl unterworfen ist, dann weiß man, daß die Folge $Z_n = \frac{1}{n}\sum_{i=1}^{n} X_i - \frac{1}{n}\sum_{i=1}^{n} EX_i$ fast sicher gegen 0 konvergiert. Sind alle EX_i gleich, nämlich $EX_i = p, i = 1,..n$, dann konvergiert die Folge Z_n fast sicher gegen die Konstante p.

Satz von Tschebyscheff: Ist $X_1, X_2, ...$ eine Folge paarweise unabhängiger Zufallsgrößen, deren Varianzen DX_i gleichmäßig beschränkt sind, d.h. $DX_i \leq C$ (eine Konstante) für alle i, so ist diese Folge dem schwachen Gesetz der großen Zahl unterworfen.

Der Satz von Tschebyscheff drückt eine hinreichende Bedingung dafür aus, daß eine Folge von Zufallsgrößen dem schwachen Gesetz der großen Zahl unterworfen ist.

Satz von Kolmogorow: Genügt die Folge X_i voneinander unabhängiger Zufallsgrößen der Bedingung $\sum_{i=1}^{\infty} \frac{DX_i}{n^2} < \infty$, so ist diese Folge dem starken Gesetz der großen Zahl unterworfen.

B.2 Diskussion

Sei:

K	Anzahl der zur Verfügung stehenden Trainingsmuster
n	Anzahl der Eingänge eines Neurons im Netz
$\vec{x}_j, \vec{y}_j$	normierte Trainingsvektorpaare, j=1,2,...,K
$\vec{w}_i$	Gewichtsvektor des i-ten Eingang eines Neurons, i=1,2,...n
X_i	eine Zufallsgröße für den i–ten Eingang eines Neurons, i=1,2,...n

Wegen der stochastischen Codierung des Trainingsvektors und der Gewichte kann der Eingang i (i=1,2,...n) eines Neurons als eine Zufallsgröße betrachtet werden, obwohl die Werte der Komponenten des Trainingsvektors und der Gewichte deterministisch sind. Das Ergebnis der skalaren Multiplikation von $\vec{x}_j$ und $\vec{w}_i$ (j=1,2,...,K) kann als Zufallsgröße X_i aufgefaßt werden. Für alle Eingänge des Neurons bekommt man eine Folge $\{X_i\}$ von Zufallsgrößen. In der weiteren Diskussion wird erläutert, daß die Folge $\{X_i\}$ unter gewissen Annahmen dem starken Gesetz der großen Zahl unterworfen ist.

Die statistische Unabhängigkeit der Zufallsgrößen wird bei der Einführung des vorgeschlagenen Verfahrens vorausgesetzt. Wegen der Normierung der Trainingsmuster und Gewichte dürfen die Varianzen aller Zufallsgrößen nicht größer als Eins sein. In diesem Sinn ist die Bedingung des Satzes von Kolmogorow erfüllt. Laut des Satzes von Kolmogorow ist die Folge $\{X_i\}$ dem starken Gesetz der großen Zahl unterworfen.

Aus der Mathematik ist bekannt, daß der Mittelwert eine gute Abschätzung des Erwartungswertes ist, wenn die Anzahl der Stichproben groß genug ist. Im vorliegenden Fall entspricht K, die Anzahl der zur Verfügung stehenden Trainingsmuster, der Anzahl Stichproben. Angenommen, daß hier ein rein binärer Fall vorliegt, dann sind alle Komponenten der Trainingsmuster entweder Einsen oder Nullen. Wenn K groß genug ist, nähert sich ihr Mittelwert nach langer Beobachtung 0,5, d.h. der Erwartungswert ist quasi 0,5. Laut Gesetz von Kolmogorow wird die Folge $\frac{1}{n}\sum_{i=1}^{n} X_i$ gegen 0,5 konvergieren. Mit anderen Worten, es pendelt der Wert von $\frac{1}{n}\sum_{i=1}^{n} X_i$ um 0,5 bei großen Werten von n. Der Term $\frac{1}{n}\sum_{i=1}^{n} X_i$ ist nach der Einführung des $\frac{1}{N}$–Mittelungsverfahrens genau die Eingabe eines Neurons. Dieses Resultat hat zur Folge, daß in großen Netzen jedem Neuron nur der Wert 0,5 zugeführt wird, wenn die Menge der Trainingsmuster ebenfalls groß ist. Bei dieser Situation läßt sich das Netz nicht mehr trainieren. Nach dieser Überlegung könnte man quantitativ herleiten, wie groß die Anzahl der Eingänge eines Neurons höchstens sein darf, damit das entsprechende Netz noch konvergiert. Wenn eine solche Abschätzung für einzelne Neuronen getroffen würde, könnte eine Aussage über die Skalierbarkeit des vorliegenden Verfahrens hergeleitet werden. Natürlich scheint die Annahme, daß der Erwartungswert aller Eingänge gleich 0,5 ist, ein Sonderfall. Jedoch kann dies im vorliegenden Verfahren leicht vorkommen, weil alle Gewichte im Netz mit Nullen (im Raum der Maschinenvariablen) initialisiert werden. Damit ist das Resultat der skalaren Multiplikation von $\vec{x}_j$ und $\vec{w}_i$ (j=1,2,...,K) Null (im Raum der Maschinenvariablen), die dem Wert 0,5 im Wahrscheinlichkeitsraum entspricht.

Die obige Überlegung entspringt der Beobachtung aus der Software–Simulation, daß alle Gewichte bei dem Wert Null (im Raum der Maschinenvariablen) steckenbleiben, wenn das Netz beim Training nicht konvergiert. Außerdem zeigt das Verfahren beim Umgehen mit Aufgabenstellungen, die eine große Anzahl von Trainingsmustern besitzen, meistens Schwierigkeiten bei der Konvergenz . Die genaue Ursache soll in der Zukunft weiter studiert werden.

C Mehrfach verwendete Formelzeichen und Abkürzungen

$\vec{a}^{[l]}$	gewichteter Eingangsvektor der Schicht l
$a_j^{[l]}$	j–te Komponente von Vektor $\vec{a}^{[l]}$
ACON	*all classes in one network*
ADD	Zählerlänge des ADDIE
ADDIE	adaptives Glied (*adaptive digital element*)
ANN	künstliches neuronales Netz (*artificial neural network*)
az	aktueller Zählerstand des ADDIE
az_0	alter Zählerstand des ADDIE
B	zufällige und jeweils zu allen anderen Bitströmen des Netzes stochastisch unabhängige Folge von Nullen und Einsen auf einer Leitung
$\bar{B}$	Negation der Folge B
$B_x, B_y, B_z...$	entsprechende Bitfolgen, die durch Codierung aus Maschinenvariablen x,y,z.. gewonnen werden
B–Funktion	Bogenfunktion (Funktion bogenförmiger Kennlinie)
BE_{z*s}	Buchstabenerkennung bei einer binären z mal s Bitmap–Darstellung
BP	Backpropagation (*back propagation*)
$D(u,v)$	Absoluter Abstand zwischen den reellen Werten u und v, d.h. $D(u,v) = \|u - v\|$
DSC	Digital–Stochastik–Codierer
E	gesamter Lernfehler über die ganze Trainingsmenge
$\tilde{E}$	durchschnittlicher Lernfehler über N_F hintereinanderfolgenden Epochen
$\bar{E}_i$	durchschnittlicher Lernfehler über Epoche i

$\bar{E}$	durchschnittlicher Lernfehler über die ganze Trainingsmenge
IND	Zählerlänge des INDIE
INDIE	integratives Glied zur Speicherung der Gewichte
iz	aktueller Zählerstand eines INDIE
iz_0	alter Zählerstand eines INDIE vor einem gewissen Zeitpunkt des Trainings
K	Anzahl der zur Verfügung stehenden Trainingsmuster
LMS	*least mean square*
lsb	*least significant bit*
M	Maschinenvariable, die durch lineare Transformation und Quantisierung aus einer Problemvariablen gewonnen werden kann und im Intervall $[-1, 1]$ liegt
M–Bereich	Wertebereich von Maschinenvariablen, d.h. Intervall $[-1, 1]$
MLN	*multilayer feedforward network(s)*
msb	*most significant bit*
n	Anzahl der Eingänge eines Neurons
n_l	Anzahl der Neuronen in der Schicht l, $l = 1, ..., N$
N	Anzahl der Schichten des Netzes
$\bar{N}$	Anzahl der Takte für die Dauer des Anlegens eines Trainingsmusters
N_F	Anzahl der hintereinanderfolgenden Epochen eines Trainingsvorgangs
NN	Neuronales Netz (*neural network*)
N_N	Anzahl der Epochen für den zwangsweisen Abbruch eines Trainingsvorgangs
N_T	Anzahl der Wiederholung eines gleichartigen Trainingsvorgangs
$\tilde{N}_E$	durchschnittliche Anzahl von Epochen, die das Training zur Konvergenz benötigt
o	Eingangssignal eines Neurons

O_{pj}	Ausgangssignal des j–ten Neurons beim Anlegen des p–ten Trainingsmusters beim Netzeingang
OCON	*one class in one network*
OCR	*Optical Character Recognition*
$P(B=1)$	Wahrscheinlichkeit für das Auftreten einer Eins in der Folge B
P_d	Parity–Problem der Dimension d
P_{max}	Normierungsfaktor für Problemvariable: Größter vorzeichenloser Betrag aller sich in einem NN–Netz befindenden Datengrößen
R	Problemvariable mit einem problemabhängigen, endlichen Wertbereich mit theoretisch idealer Genauigkeit
r_i	reelle Zufallszahlen, die im Interval [-1,1] liegen, $i = 1, 2$
R_n	Bit–Anzahl des aktuellen Bereichs im Schieberegister für die neue S–Funktion
R_t	Schwelle für die neue S–Funktion
RUN	Runlänge des Squashfunktionsautomaten, welche die Steilheit der Squashfunktion bestimmt
RUNAB	Runlänge des B–Funktionsautomaten, welche den Verlauf der B–Funktion steuert
$s(.)$	Sigmoidfunktion $s(x) = \frac{1}{1+e^{-\mu x}}$
$s'(.)$	Ableitung der Sigmoidfunktion $s'(x) = \mu s(x)[1 - s(x)]$
S–Funktion	Squashfunktion, die eine S–förmig gekrümmte, monoton steigende Kennlinie hat
$\vec{S}_k$	Vektor für die Suchrichtung bei der k–ten Iteration
$\vec{S}_k(p)$	Vektor für die Suchrichtung der k–ten Iteration beim Anlegen des p–ten Trainingsmusters
$s_{ij}^k(p)$	Komponente des Vektors $\vec{S}_k(p)$
$\vec{t}$	Soll–Vektor aus der Trainingsmenge
t	theoretisches Resultat einer Berechnung
t_c	Taktperiode

t_{CA}	Taktfrequenz, mit welcher der Zählerstand für die Decodierung eingelesen wird.
t_j	j–te Komponente von Vektor $\vec{t}$
T	Anzahl Takte
ULSI	*ultra-large-scale integration*
VLSI	*very-large-scale integration*
W–Bereich	Wertebereich der Wahrscheinlichkeit, nämlich das Intervall $[0, 1]$
W_r	Stärke des Rauschanteils für die Gewichte
$\hat{w}_{ij}$	Gewichte, die durch P_{max} normiert wurden
$\vec{W}$	Vektor der Dimension $\Sigma(\vec{W} \in R^{\Sigma})$, welcher alle Gewichte des Netzes umfaßt.
$\vec{w}_i$	Gewichtsvektor des i–ten Eingangs eines Neurons, i=1,2,...n
$w_{ij}^{[l]}(k)$	von der k-ten Iteration gewonnenes Gewicht an der Leitung zwischen Neuron i in der Schicht $l-1$ und Neuron j in der Schicht l
X_r	Stärke des Rauschanteils für die Eingaben
$\bar{x}$	Stichprobenmittelwert einer Zufallsvariablen
$\bar{X}$	Zufallsvariable
$\tilde{X}$	Abschätzung einer Variablen X im Intervall $[0, 1]$
$\vec{x}^{[l]}$	Aktivierungsvektor der Schicht l; $\vec{x}^{[0]}$ =Eingangsvektor $\vec{u}$ des Netzes
$x_j^{[l]}$	j–te Komponente von Vektor $\vec{x}^{[l]}$
$\vec{x}_j, \vec{y}_j$	normierte Trainingsvektorpaare, j=1,2,...,K
X_i	Zufallsgröße
α	Momentum-Konstante
δ	lokaler Fehler eines Neurons als Produkt der Ableitung der Sigmoidfunktion und dem summierten gewichteten Fehler der nachgeschalteten Neuronen
$\Delta w_{ij}^{[l]}(k)$	Gewichtsänderung für $w_{ij}^{[l]}(k)$ bei der k-ten Iteration

$\Delta w_{ij}^{[l]}(k+1,r)$	Gewichtsänderung für $w_{ij}^{[l]}(k)$ bei der Präsentation des r-ten Musterpaares während der k-ten Iteration
$\nabla E_p(\vec{W}_k)$	Gradient der k–ten Iteration beim Anlegen des p–ten Trainingsmusters
γ	Lernrate
μ	Steilheit der Sigmoidfunktion
$\tilde{\mu}$	Steilheit der Sigmoidfunktion für den Fall, daß das $[\frac{1}{N}]$–Problem auftritt
$\hat{\mu}$	Steilheit der Sigmoidfunktion für den Fall, daß die [0,1]–Einschränkung auftritt
σ	Standardabweichung
Σ	Anzahl sämtlicher Gewichte in einem Netz
σ_{max}	maximale Standardabweichung, welche in der Mitte des W–Bereiches gewonnen werden kann
$\theta_{\vec{x}}(0,1)$	**Maximale Abweichung** des Vektoren $\vec{x}$ vom Wertebereichsrand. $$\theta_{\vec{x}}(0,1) = 2 \max_i (\min_{j=0,1} D(x_i,j)) \qquad \text{(C.1)}$$
ϵ	vorgegebene positive Konstante, die beliebig klein werden kann.

Lebenslauf

Name		Zhu
Vorname		Liyun
Geburtstag		08. November 1957
Geburtsort		Beijing, China
Familienstand		verheiratet, zwei Kinder
Schulbildung	1966-1972	Grundschule in Chengdu, China
	1972-1976	Mittelschule in Chengdu, China
	1990-1991	Sprachkurs am Goethe-Institut, Beijing
	1991-1992	Sprachkurs am Goethe-Institut, München
Studium	1978-1982	Studium der Mathematik an der Xian–Jiaotong Universität Fachrichtung Computer–Mathematik Abschluß „Bachelor of Science“
	1985-1988	Studium der Angewandten Mathematik am „Chengdu College of Geology“ Fachrichtung, Geomathematik Abschluß „Master of Science“
	1993-1994	Aufbaustudium an der Mathematisch-Naturwissenschaftlichen Fakultät der CAU Kiel Fachrichtung: Geophysik
Berufstätigkeit	1976-1978	Techniker im Rechenzentrum des „Chengdu College of Geology“
	1982-1985	Dozent an der Fakultät für Angewandte Mathematik am „Chengdu College of Geology“ Geleisteter Unterricht: „Datenverarbeitung für Geophysiker“ „Computersprachen für Programmierer“ „Komplexe Funktionen und Vektor-Analysis“ „Statistische Mustererkennung“
	1988-1990	Systemadministrator am Rechenzentrum des „Chengdu College of Geology“
	seit 1995	Doktorand und wissenschaftliche Hilfskraft an der Professur Technische Informatik der Universität der Bundeswehr Hamburg